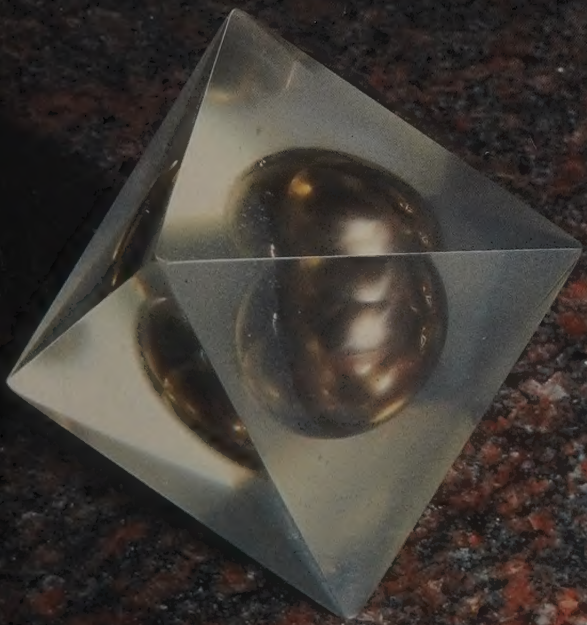
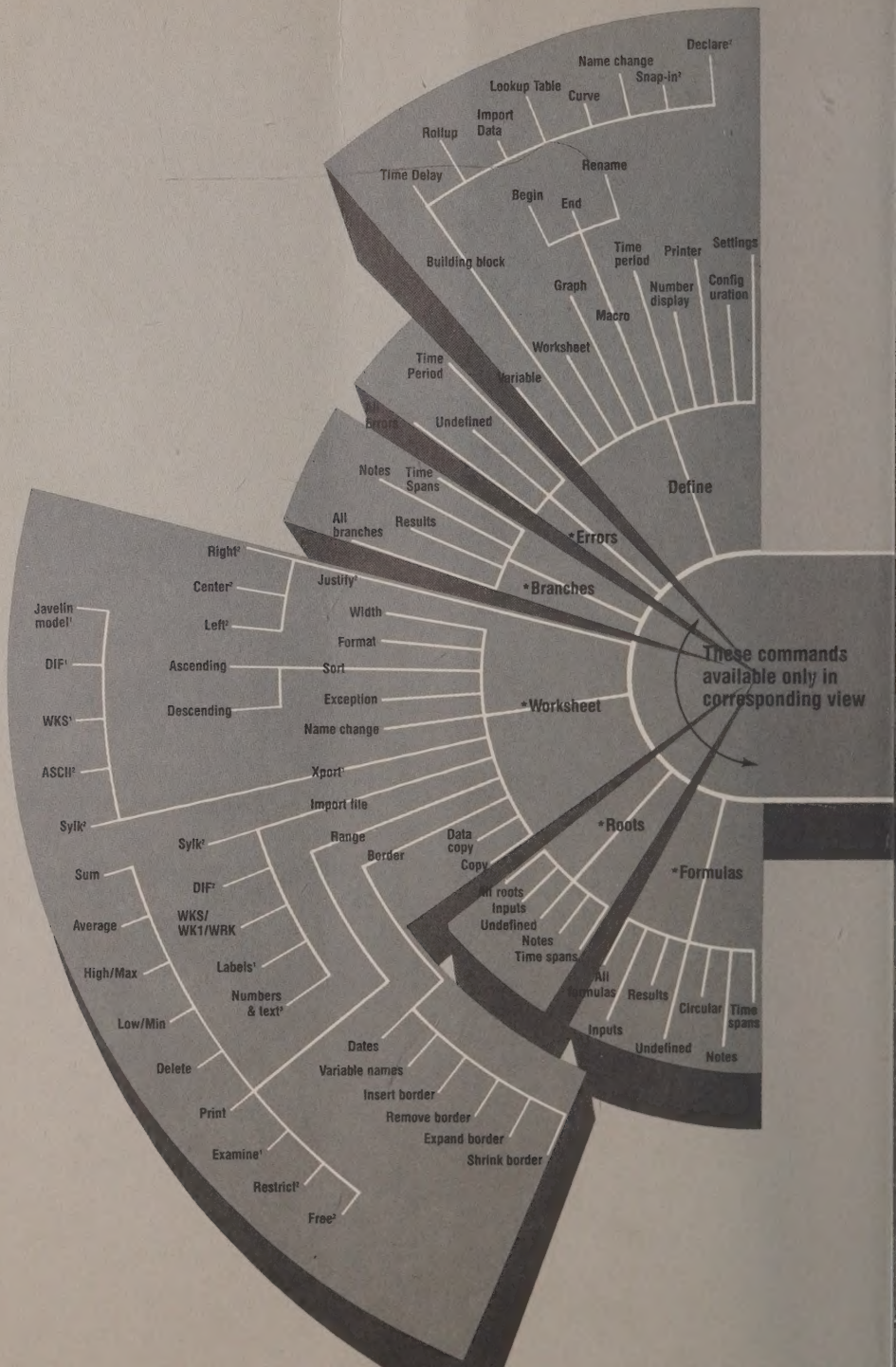
SYBEX

UNDERSTANDING JAVELIN PLUS™

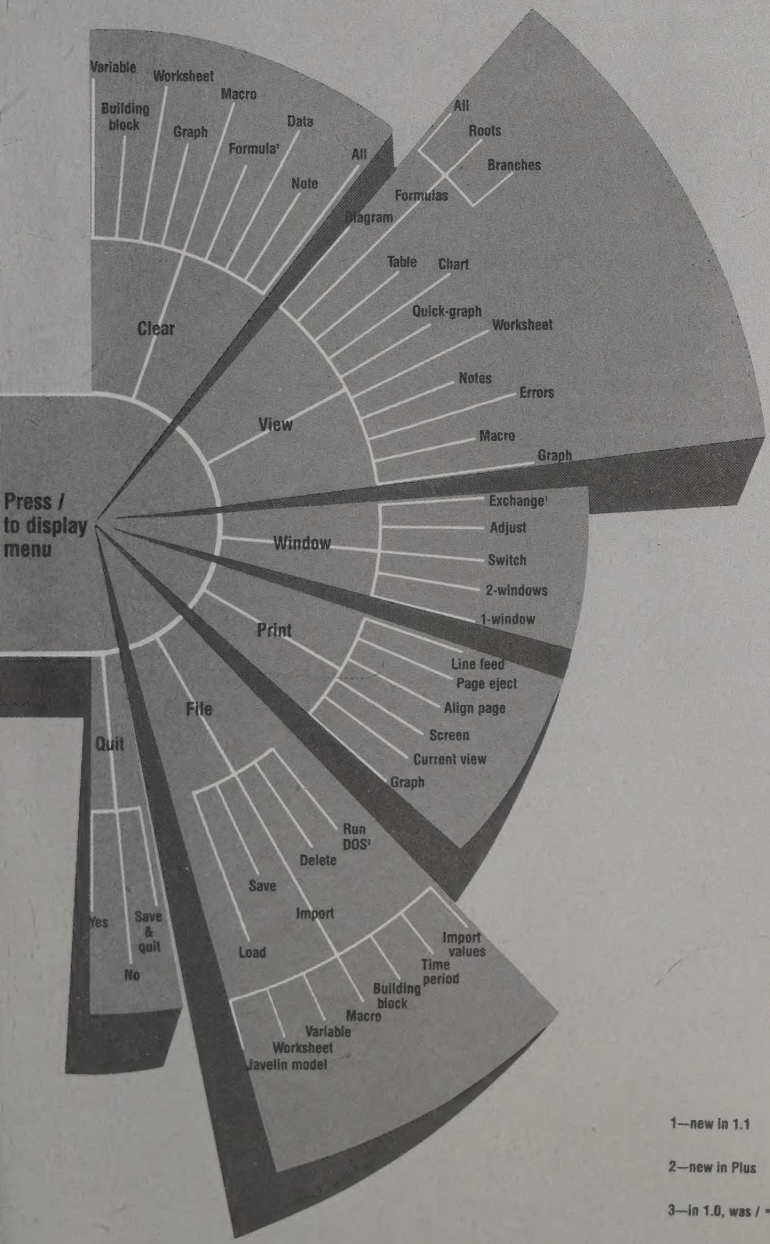
John R. Levine, Margaret Levine Young, Jordan M. Young II



SYBEX


Declare²
Name change
Snap-In²
Lookup Table
Curve
Import Data
Rollup
Time Delay
Rename
Begin
End
Time period
Printer
Settings
Building block
Graph
Number display
Config uration
Macro
Worksheet
Time Period
Variable
Undefined
Notes
Time Spans
All branches
Results
Define
*Errors
*Branches
These commands available only in corresponding view
Right²
Center²
Justify²
Left²
Javelin model¹
Width
Format
DIF¹
Ascending
Sort
Descending
Exception
*Worksheet
WKS¹
Name change
ASCII²
Xport¹
Import file
Sylk²
*Roots
Sylk²
Range
Data copy
Border
Copy
Sum
*Formulas
DIF²
All roots
Inputs
Undefined
Average
WKS/ WK1/WRK
Notes
Time spans
High/Max
Labels¹
All formulas
Results
Circular
Time spans
Numbers & text²
Low/Min
Inputs
Undefined
Notes
Dates
Delete
Variable names
Insert border
Print
Remove border
Expand border
Examine¹
Shrink border
Restrict²
Free²

JAVELIN MAIN MENU



1—new in 1.1

2—new in Plus

3—in 1.0, was / *Worksheet Import Text

Ted -
Ah - would that's sometimes
host an accolade, I hope you
recognize the fruits of some seeds that
you planted long ago, as
they are scattered through
this book.
With deep appreciation

+ Mary

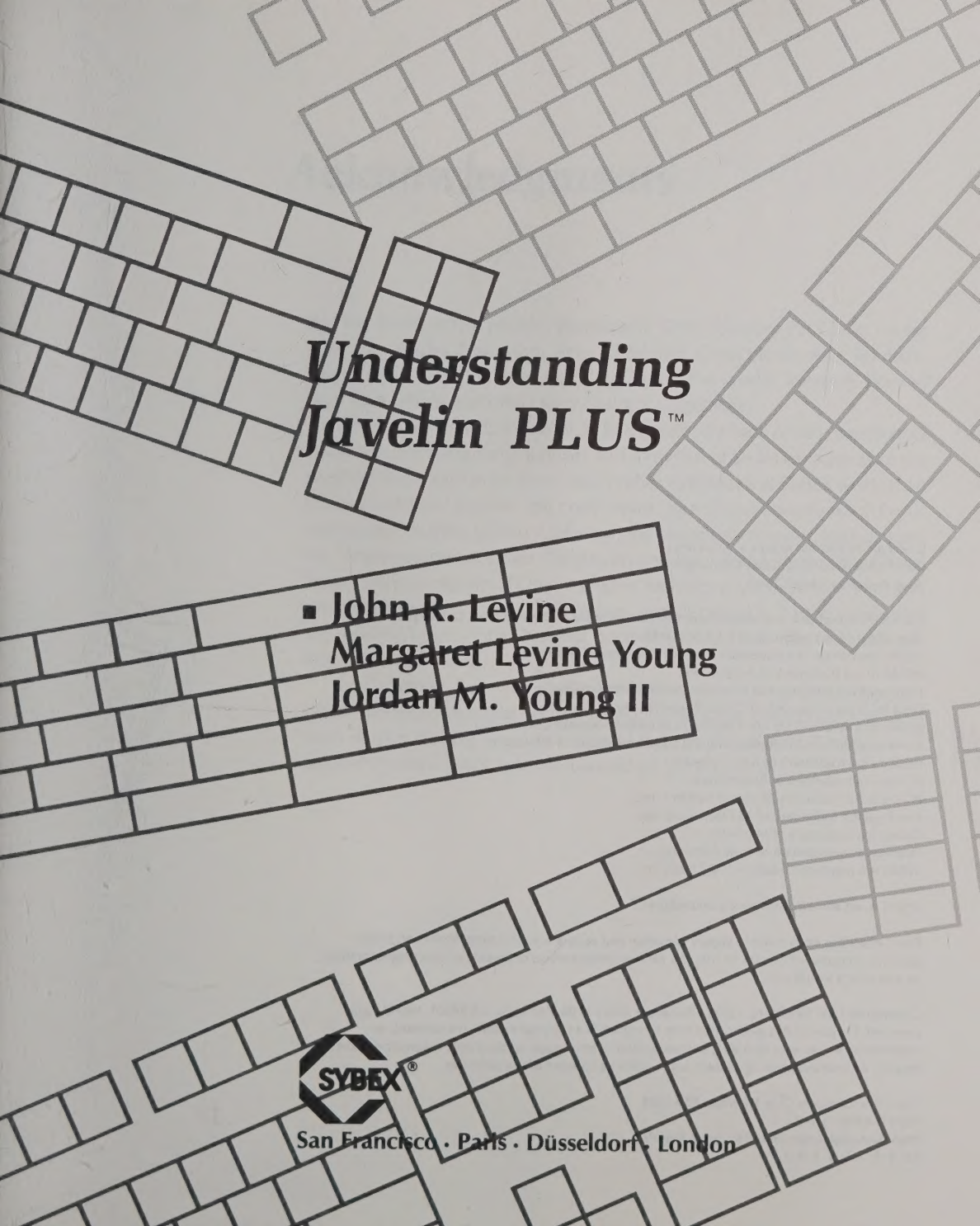
+ John
(5-see p. VII

Understanding Javelin PLUS

[Sent by Lydia,
1 Apr 88

Understanding Javelin *PLUS*™

John R. Levine
Margaret Levine Young
Jordan M. Young II



SYBEX®
San Francisco · Paris · Düsseldorf · London

Cover art by Thomas Ingalls + Associates
Cover photography by Casey Cartwright
Book design by Ingrid Owen

1-2-3, DIF, Symphony, and VisiCalc are trademarks of Lotus Development Corporation.
Aboveboard is a trademark of Intel Corporation.
Apple LaserWriter is a trademark of Apple Computer Corporation.
dBASE III is a trademark of Ashton-Tate.
FreeLance is a trademark of Graphics Communications, Inc.
GEM Paint is a trademark of Digital Research, Inc.
Javelin and Javelin PLUS are trademarks of Javelin Software Corporation.
MS-Windows Paint, Multiplan, and SYLK are trademarks of Microsoft Corporation.
Paradox is a trademark of Ansa Software.
PC-File is a trademark of ButtonWare.
PC-Paint is a trademark of Mouse Systems, Inc.
RamPage is a trademark of AST Research, Inc.
R:Base is a trademark of Microrim.
Sideways is a trademark of Funk Software.
SYBEX is a registered trademark of SYBEX, Inc.

SYBEX is not affiliated with any manufacturer.

Every effort has been made to supply complete and accurate information. However, SYBEX assumes no responsibility for its use, nor for any infringements of patents or other rights of third parties which would result.

Copyright©1987 SYBEX Inc., 2021 Challenger Drive #100, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

Library of Congress Card Number: 87-61202
ISBN 0-89588-358-9
Manufactured in the United States of America
10 9 8 7 6 5 4 3 2 1

Acknowledgments

■ We thank many people, particularly Grace Carney, Pat Clark, Louise Cousins, Charles Frankston, Arye Gittelman, Chris Herot, Alec Johnson, Peter Pathe, David Phoenix, and Greg Rigby, all at Javelin Software. They all read many drafts and made many helpful suggestions.

Our editor Fran Grimble of SYBEX graciously took on the unenviable task of coordinating three authors and a myriad of technical aspects of the book, all from 3000 miles away. Many other individuals at SYBEX worked on this book: Jeffrey Bartlett, the copy editor, turned our sometimes dubious writing into English. Clifton Anderson, the technical reviewer, kept us honest. Word processors Olivia Shinomoto and John Kadyk, typesetter Cheryl Vega, graphic artist Ingrid Owen, graphic technician Michelle Hoffman, and proofreader Jeffrey Green each played an indispensable part in producing the finished book you're reading now. We would also like to thank our indexer, Josh Bernoff.

Lydia Spitzer, Beth Bryant, Jan Brink, Katy Weeks, and David Kay provided invaluable personal support without which this book could not have been written. Finally, we owe a special debt to Theodor H. Nelson, who assured us many years ago that it would all come to pass.

Table of Contents

Introduction

- A Short History of Modeling xxiv
  - Versions of Javelin xxv
    - Javelin 1.0 xxv
    - Javelin 1.1 xxv
    - Javelin PLUS xxvi
  - Organization xxvi

CHAPTER ONE

What Is a Model?

- Differences Between Javelin and Lotus 1-2-3 2
  - The Parts of a Model 3
  - How to Design a Model 4
    - The Idea of the Model 4
    - What Is a Variable? 5
    - How to Name Variables 6
    - How to Design a Model 7
    - How to Make Assumptions 8
    - Summary: The Model's Variables and Formulas 9
  - Good Modeling Practices 9
    - Plan Ahead 9
    - Design from the Top Down 10
    - Make Models the Right Size 12
    - Base Models on Decision Items 12
    - Model Flows Rather than Stocks 12
    - Choose Appropriate Time Periods 13
  - Conclusion 13

CHAPTER TWO

Javelin Basics

- Parts of the Javelin Screen 16
  - The Edit Line 16
  - The Information Line 17
  - The Status Line 18
  - The View Area 18
  - The Cursor 20
- The PC Keyboard and Javelin 20
  - The Function Keys 20
  - The Typewriter Keys 21
  - The Numeric Keypad 24
- Two Basic Function Keys 25
  - The Help Key 25
  - The Calc Key 26
- Javelin Commands 27
  - The Main Menu 28
  - How to Choose a Command from a Menu 28
  - Options Menus 28
  - How to Back Out of a Command 28
  - More Options 30
  - Memory in Menus 30
- How to Quit Javelin 30
- Conclusion 32

CHAPTER THREE

Making a Model

- Entering Formulas 34
- Viewing a Table of Variable Values 36
- Entering Data 38
  - Creating Another Time-Series Variable 38
- Looking at the Answer 40
- Creating a Report 41

Creating a Worksheet 41
Creating Another Worksheet 43
Saving the Model 48
Printing a Worksheet 48
Conclusion 49

CHAPTER FOUR

Managing Files and Printing Views

■ File Operations 52
File Names 52
Path Names and Subdirectories 53
Javelin's Default Directory 55
Using Javelin's / File Commands 55
Loading a File from the Disk 55
Possible Problems Loading a File 56
Saving a Model on a Disk 57
Saving a File a Second Time 57
Clearing a Model in Memory 58
Possible Problems Saving a File 59
Deleting Files 60
Running DOS from within Javelin PLUS 60
Printing Information from a Model 61
Aligning the Top of the Page 61
Printing the Current View 62
Printing the Contents of the Screen 63
Moving the Paper 64
Quitting Javelin 64
Conclusion 64

CHAPTER FIVE

Special Keys and Settings Sheets

■ Editing Keys 66
The Edit Key 66

Cue Movement, Insertion, and Deletion 67
The Goto Key 69
Going to a Time Period 70
The Spell Key 71
The Spell Key and Word Order 72
Settings Sheets 73
Using Settings Sheets 73
Value Settings 75
Multiple-Choice Settings 76
Toggle Settings 76
The Model and Configuration Settings Sheets 77
Conclusion 80

CHAPTER SIX

Javelin Views and Windows

■ Controlling Windows: The / **W**indow Command 82
Moving the Cursor Between Windows 82
Moving the Window Divider 83
Swapping Two Windows 83
Choosing One or Two Windows 84
Looking in Windows: The / **V**iew Command 84
The Diagram View 85
Printing from the Diagram View 89
The Formulas View 89
Viewing All Formulas 89
Viewing the Roots of a Variable 91
Viewing the Branches of a Variable 93
The Table View 95
The Chart View and the Quick-graph View 98
Entering and Editing Data 101
The Notes View 102
The Other Views 105
The Worksheet View 105
The Graph View 105

The Errors View 106
The Macro View 106
The Sync Key 106
Conclusion 107

CHAPTER SEVEN

Defining Variables and Entering Data Using Worksheets

■ Uses of the Worksheet 110
Note to 1-2-3 Users 111
Parts of the Worksheet 111
Displaying the Worksheet 113
Moving Around the Worksheet 114
Defining Input Variables in the Worksheet 114
Entering Dates in a Border 115
Defining Variables in the Worksheet 116
Entering Values in the Body of the Worksheet 117
Inserting and Deleting Parts of a Worksheet 119
Deleting Rows and Columns 119
Inserting Rows and Columns 121
Defining Constants in the Worksheet 121
Correcting Mistakes in the Worksheet 122
The Clear Variable Command 123
Changing a Variable Name in the Worksheet 123
Defining Calculated Variables in the Worksheet 126
Using Indented Borders 127
Creating New Variables with / Worksheet Copy 129
Using the / Worksheet Copy Command 129
Copying Border Cells 130
Copying Formulas 132
Exception Cells 133
Entering Label Exception Cells 134
Entering Variable Value Exception Cells 134

Highlighting Exception Cells 135
Deleting and Copying Exception Cells 136
Creating Multiple Worksheets 136
Filling in Border Dates Automatically 136
Indented Borders and Variable Names 137
Entering Constant Variables into Indented Borders 138
Converting Dates in the Worksheet 139
Formatting Variables, Worksheets, Rows, Columns, and Cells 141
Number Formats 141
Setting the Model Format 143
Setting a Variable's Format 144
Setting Worksheet, Border, and Cell Formats 146
Conclusion 148

CHAPTER EIGHT

Formulas and Functions

■ The Sample Model: ECON 150
Using / **D**efine **V**ariable to Create Variables 151
Creating the ECON Model 152
Using the Variable Settings Sheet 153
Typing in Numbers 153
Numeric Formats 156
International Variations on Numeric Formats 158
The Point Key 159
Order of Precedence 163
What Is a Function? 164
Results of Functions 166
Comparing Javelin's Functions to 1-2-3's 166
Types of Functions 166
Basic Functions 173
Logical Functions 177
The Place-Holder Function 180
Conclusion 180

CHAPTER NINE

Javelin and Time

- The Javelin Calendar 182
  - Typing in Dates 182
  - Yearly Dates 183
  - Quarterly Dates 183
  - Weekly Dates 183
  - Dates with Custom Periods 184
- The Sample Model: Mr. Pizza 184
- Time Span of the Model 184
- Time Spans of Variables 186
  - Input Variables 187
  - Calculated Variables 188
- The ; Operator 189
- Time-Handling Functions 193
- Time-Shifting Functions 193
- Time Conversion 194
  - Automatic Time Conversion on a Worksheet 194
  - Time-Conversion Functions 194
  - Splitting Larger Time Periods 196
  - Combining Smaller Time Periods 198
  - Combining Weeks into Larger Periods 198
- Fiscal Years 201
  - Continuing the Mr. Pizza Model 202
- Accumulation Functions 203
- The @ Operator and the VALUE Function 206
- Extrapolation and Interpolation Functions 207
  - Extending a Time Series Using Extrapolation 208
  - Filling Gaps in a Time Series Using Interpolation 209
- Creating a More Complex Cash Flow Model 209
- Functions That Count Days in Larger Periods 211
  - Continuing the Enhanced Mr. Pizza Model 213
  - Adding a Few More Changes 215
- Time-Span Limiting Functions 217
- Smoothing Functions 219

Rate Functions 221
Date Arithmetic Functions 222
Conclusion 224

CHAPTER TEN
Custom Time Periods

■ Time-Related Custom Periods 226
Nontime Custom Periods 227
Defining a Custom Period 227
A, B, and Record Periods 227
First Example: 5-4-4 Production Calendar 228
#NODATE: When Javelin Can't Convert Between Periods 229
Defining a Nontime Custom Period 230
Defining a Time-Related Custom Period 230
Converting To and From Custom Periods 232
Custom Period Conversion Functions 233
Other Time Functions Used with Custom Periods 233
Employee Model for 401K Plans 234
The Custom Period: Emp 234
Input Variables for 401K Model 235
Calculated Variables for 401K Model 235
Checkbook Model 240
First Custom Period: CK 240
Input Variables for Checkbook Model 242
Second Custom Period: Dep 242
Calculated Variables for Checkbook Model 243
Defining One Custom Period in Terms of Another 245
Conclusion 246

CHAPTER ELEVEN
Special Values and Diagnosing Problems

■ General Problems with Calculated Variables 248
No Answers Calculated 249

Answers for Wrong Dates 249
Special Values 249
Wrong Answers 249
Diagnostic Tools 250
The Errors View 250
The Roots and Branches Views 251
Time-Span Problems 251
Mismatched Input Time Spans 252
Other Time-Span Problems 254
Fixing Time-Span Problems in Javelin 1.1 256
Time Spans in Javelin PLUS 257
Time Span Problems in Spiral Models 257
Circular References 258
A Compensating Balances Model 259
Special Values 260
#ARITH: Arithmetic Error 261
#CALC: Not Yet Calculated 262
#CIRC: Circular Reference 263
#ERROR: Formula Too Complicated 262
#INF: Infinity 262
#NA: Not Available 262
#NODATE: Date Needed for Variable 264
#OUTMEM: Out of Memory 264
#RANGE: Out of Time Range (Javelin 1.1) 264
#BLANK: No Value Assigned (Javelin PLUS) 265
#UNDEF: Undefined Variable 265
Conclusion 266

CHAPTER TWELVE
Graphing

■ The Sample Model: ECON 268
Hardware Requirements 268
Using Graphs 269
Creating a Graph 269

The Graph Settings Sheet 271
Displaying a Graph 272
Editing a Graph 274
Scaling a Graph 274
Legends and Labels 274
Numeric Formats and Dates 275
The Line Graph 276
Right Y-Axis 277
The Bar Graph 278
The Stacked Bar Graph 280
The Percent Bar Graph 281
The Horizontal Bar Graph 283
The Pie Chart 286
The XY Scatter Plot 288
High-Low-Close Graphs (Javelin PLUS) 292
Printing a Graph 293
Tips on Creating Graphs 295
Conclusion 296

CHAPTER THIRTEEN

Advanced Worksheets and Printing

■ The Sample Model: Budget 298
The Worksheet Settings Sheet 301
Column Widths 303
Worksheet Range Calculations 304
Defining a Worksheet Range Calculation 305
Looking at a Worksheet Range 307
Deleting a Worksheet Range Calculation 307
Advantages and Disadvantages 307
Copying Formulas and Data 308
Copying Formulas and Input Data 309
Creating an Input Variable from a Calculated Variable 310
Renaming Groups of Variables 311
Using the Corner Border 313

Multiple Borders 314
Inserting Another Border 315
Inserting Horizontal Borders 317
Removing a Border 318
Inserting Borders for Title Rows 318
Removing the Top Border 319
Expanding and Shrinking the Top Border 320
Superheadings 321
Filling in Variable Names Automatically 322
Clearing the Worksheet 322
Filling in Selected Variable Names 322
Reviewing the 401K Worksheet 324
Sorting Rows 325
Fine-Tuning the Appearance of the Worksheet 327
Scaling Numbers in a Worksheet 327
Label Prefixes 328
Controlling the Appearance of Borders 330
Deleting a Range 331
Printing a Range 331
The Printer Settings Sheet 332
Using Setup Strings to Control the Printer 332
Page Limits and Other Settings 335
Headers and Footers 335
Saving Printer Settings 338
Conclusion 338

CHAPTER FOURTEEN
Macros

■ Macro Basics 341
Recording a Macro 342
Running a Macro 342
The Macro View 343
Macro Commands for Keystrokes 343
Debugging and Explaining Macros 344

Single-Stepping Execution 345
Documenting Macros 346
Interactive Macros 347
Accepting a Single Piece of Information 347
Pausing for Full-Screen Data Entry 348
Variable Information in Macros 349
The Histogram Model 350
Conditional Processing 355
Controlling the Order of Execution 357
Running One Macro from Within Another 358
More Automation in the Histogram Model 359
Calling In Another Model 359
Macro Programming 360
Macro Menus 361
Controlling the Screen 361
Controlling the Keyboard 362
The Auto-Executing Macro 362
The Worksheet Range Restrict Command 363
A Complete Application: The Amortization Macro 364
Good Macro Practices 368
Make Your Macros Robust 369
Make Your Macros Readable 369
Protect Your Model 370
Notes on the @PAUSE Command 370
Conclusion 371

CHAPTER FIFTEEN
Statistics

■ Least-Squares Linear Regression Analysis 374
Predicting the Value of One Series Based on Another 375
Calculating the Correlation and the Equation of the Line 378
Predicting the Value of One Series Against Time 380
More Sophisticated Analysis 382
Analysis of Variance 384

Random Numbers 386
Trigonometric Functions 388
Conclusion 389

CHAPTER SIXTEEN
Financial Calculations

■ Present and Future Values 392
Other Present Value Calculations 395
The Sample Model: FUND 397
Getting IRR to Converge 399
Depreciation 400
Half-Year Depreciation 403
Bond Pricing and Yields 404
Determining Numbers of Years for Bond Functions 405
Conclusion 407

CHAPTER SEVENTEEN
Building Blocks

■ Introducing Javelin's Building Blocks 410
The Time Delay Building Block 412
Creating a Building Block 414
Connecting a Building Block to Other Variables 415
Selecting the Period 417
Editing the Time Delay Curve 418
Rescaling the Delay Curve 420
Viewing, Editing, Renaming, and Deleting 420
Using Time Delays for Product Rollouts 422
The Lookup Table Building Block 424
Copying a Building Block 433
The Sample Model: TAX 434
The Curve Building Block 435
Snap In Building Blocks 439
Declaring a Snap In 439

Defining a Snap In 440
The What If Snap In Building Block 441
Using What If on Mr. Pizza 442
Conclusion 444

CHAPTER EIGHTEEN

Transferring Information Between Models

■ The Import Data Building Block 447
Selecting Variables to Import 447
Improvements in Javelin PLUS 449
Performing the Import 450
Budget Consolidation Using Import Data 452
The Rollup Building Block 452
Creating the Org Chart 454
Selecting the Variables 456
Improvements in Javelin PLUS 457
Reducing the Number of Output Variables 458
Performing the Rollup 459
Commands to Import Components of Models 461
Copying Worksheets, Macros, and Building Blocks 462
Exporting a Worksheet as a New Model 463
Conclusion 463

CHAPTER NINETEEN

Using Javelin with Other Programs

■ Data Formats 466
ASCII Text Format 467
ASCII Delimited Format 468
Importing Data 468
Importing ASCII Files 469
Importing Lotus Spreadsheet Files 470
Importing Dates from a Lotus Spreadsheet 474
Importing the FUND Model from 1-2-3 475

Importing the Checkbook Model from 1-2-3 479
Importing DIF and SYLK Files 484
Exporting Data 484
Printing to a File 486
Conclusion 486

CHAPTER TWENTY

Strings and Dates

■ Text Values: Strings 490
String Operators 492
String Functions 496
Using Part of a String 497
Comparing, Finding, and Replacing 498
Using Character Codes 499
Other String Functions 500
Date Values 501
Date Operators 502
Date Functions 503
Today's Date 503
Date a Series Begins or Ends 505
First Period in a Larger Period 506
Converting Among Dates, Text, and Numbers 508
Determining Types of Values 509
Conclusion 510

CHAPTER TWENTY-ONE

Javelin Databases

■ Lists and Databases 512
Maintaining a List 514
Editing Existing Records 515
Searching a List 515
Entering New Records 517
Deleting Records 519

Sorting a List 519
Selecting Records by Criterion 522
Printing Reports 523
Relating Two or More Lists 525
The Sample Model: BROKER 525
Relating Two Lists Using INDEX 529
Looking Up Values in a Related List 529
Creating a Report Using Related Variables 531
Creating Columnar Reports from Related Lists 532
Cross-Tabulation Functions 535
Holidays and Weekends 538
Conclusion 540

APPENDIX A
Function Keys and Control Key Combinations 544

APPENDIX B
List of Sample Models 546

Index 548

Introduction

■ Our goal in this book is to introduce you to computer modeling and at the same time teach you to build models in Javelin. Starting with the basics and continuing through the details of application development, this book is intended for both novice Javelin users and experienced Javelin developers. If you have never created a computer model, the early chapters take you through that process, step-by-step. For experienced computer model makers (including Lotus 1-2-3 users, although you may not think of yourselves that way), we cover in depth Javelin's model building tools.

A Short History of Modeling

■ People have been doing business modeling with pencil and paper for centuries, but the modeling art came into its own when computers became available. Users of mainframe computers can build models with packages such as Express and IFPS, two modeling languages whose designs influenced Javelin's.

On microcomputers, the first business models were small programs written in BASIC. These models worked, but BASIC programming is rather difficult and so either models remained small or the modelers became dedicated, patient programmers. VisiCalc, the first computerized spreadsheet, made modeling possible for people who did not have the time or interest to learn to program in BASIC. Small spreadsheet models were quite practical in VisiCalc, but business users quickly found its limits.

Lotus 1-2-3 addressed many of those size restrictions. But because the program does not provide specific tools to organize models, larger 1-2-3 spreadsheets can quickly become unwieldy and hard to understand and audit. Javelin is designed specifically for modeling and provides tools that make large models practical, while its powerful Worksheet view turns out small "quick and dirty" models as easily as 1-2-3.

Versions of Javelin

■ Most of this book applies equally to Javelin 1.1 and Javelin PLUS. Some features, notably the string and database features explored in Chapters 20 and 21, are available only in PLUS.

Javelin Software started writing Javelin in the summer of 1984, when the "microcomputer revolution" was in full swing. As a result, Javelin takes full advantage of the extensions made to the original personal computer architecture. Javelin also incorporates the advances in understanding how people work with personal computers; the menus and help available are as much an improvement over 1-2-3 as 1-2-3 was over VisiCalc.

The first version of Javelin (1.0) was introduced in October 1985. Version 1.1 followed in the summer of 1986, and PLUS in the summer of 1987.

■ *Javelin 1.0*

Javelin 1.0 has been completely superseded by 1.1, although any model built in 1.0 should (and as far as we can tell, will) run in 1.1. Anyone still using 1.0 should upgrade to 1.1 or PLUS by contacting Javelin Software.

■ *Javelin 1.1*

Version 1.1 made many small improvements to 1.0. It increased the maximum model size from 1000 to 16,000 variables. It added two new functions for generating random numbers and enabled some existing functions to handle cases they could not previously, like variable interest rates in the loan payment function. New worksheet commands make it easier to create and print worksheets exactly as you want them.

Javelin 1.1 was the first version to be translated into foreign languages; it is available in German and French editions in Europe. Its currency formatting options and country drivers make Javelin effective in countries other than the United States and in languages other than English.

For developers, new macro commands were added to make it easier to create self-contained applications that run without requiring a user to know anything about Javelin.

There were also some improvements in usability; most notably, while 1.0 was copy protected, 1.1 is not. Version 1.1 makes better use of expanded memory so that models can be up to eight megabytes in size if enough memory is physically installed in the computer.

■ *Javelin PLUS*

Javelin PLUS is a major improvement over 1.1. The most important additions are text (string) and date variables, which let you create models and databases that were impossible in 1.1. New functions help you use string and date data, both in a database and as part of mostly numeric models. Rules for determining the time spans of Javelin variables were changed to make it easier to create and calculate models in which some data are missing. Snap In building blocks allow Javelin Software and others to extend Javelin for particular problems and applications.

There are also minor improvements. More file formats are supported for file import and export, notably SYLK and comma-delimited ASCII. There are more options for printing. You can run DOS commands from within Javelin. The model rollup and import facilities now handle large models conveniently. Trigonometric functions help users with engineering calculations. The entire list of changes is too long to enumerate here; for details, see the *Javelin PLUS Reference Manual*.

Throughout the book we'll point out features available only in specific versions of Javelin.

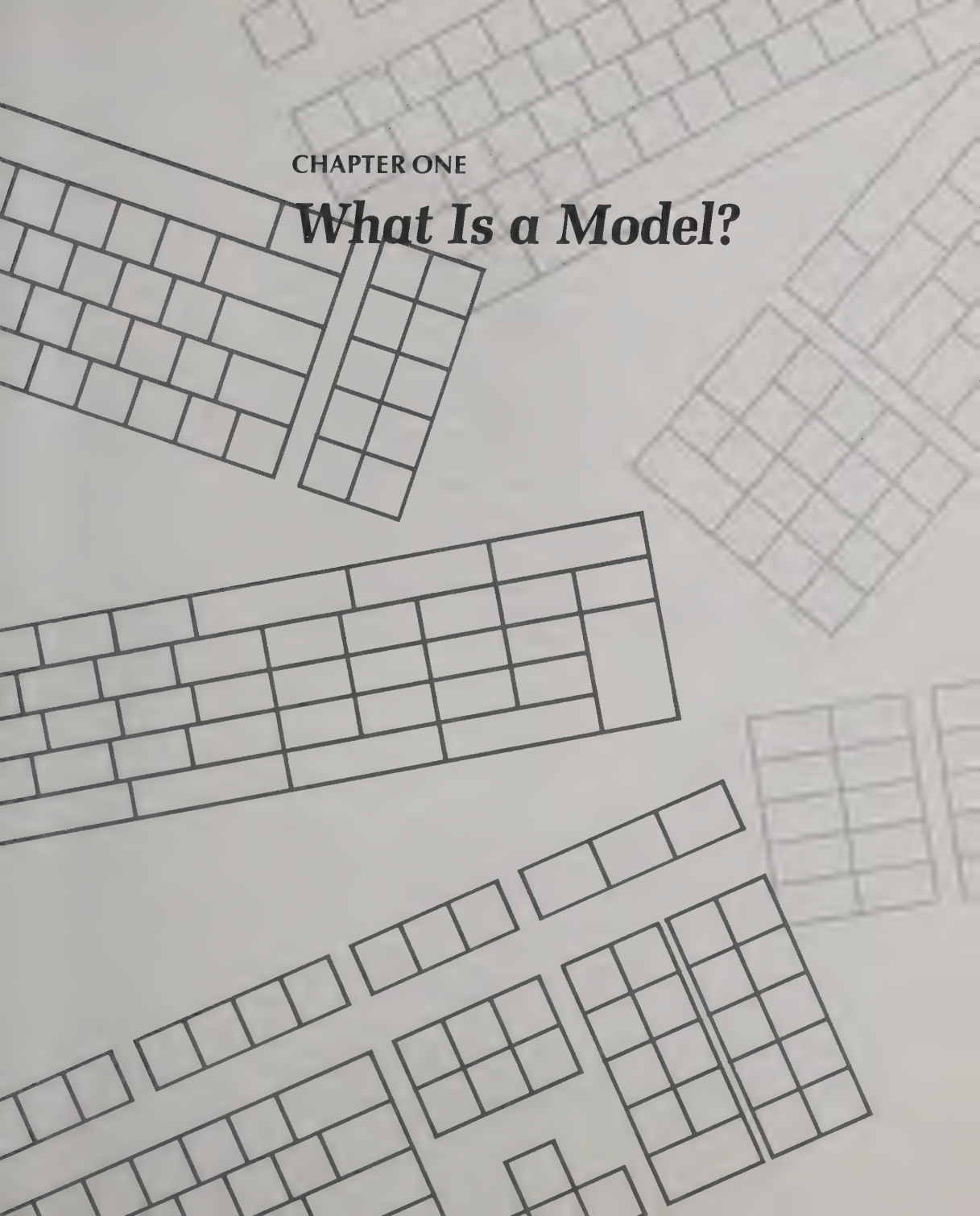
Organization

■ The first ten chapters introduce Javelin, showing you its features keystroke-by-keystroke so you learn the program as you build your first models. Be sure to read Chapter 1, which explains what a Javelin model is and gives some important hints on how to design your models. Other early chapters show how to build a model, how to give Javelin commands, and how to print and save models. The important Chapters 6 and 7 describe how the ten Javelin views work and how to use the most important one, the Worksheet view. You'll be able to use Javelin for quick and dirty calculations as well as full-blown models.

The later chapters explain more advanced capabilities. We assume that you will become familiar enough with Javelin that you will not need or want

every keystroke described. So we'll go into larger models and examples, and concentrate on what is happening rather than on the mechanics of typing. Throughout, we explain all new commands, keys, and functions as they are introduced. You'll find out about advanced features like built-in functions, graphing, building blocks, macro programming, and importing and exporting data. The last two chapters describe the most interesting features of the new Javelin PLUS—string and date variables and database functions.

We have tried continually to include a mix of fully developed models that illustrate how Javelin features fit together, along with model fragments that do things we think you'll want to include in your own models. We hope that you find this book helpful not only in learning the mechanics of using Javelin, but also in learning to design useful, powerful models.



CHAPTER ONE

What Is a Model?

In this book, we will talk about how to design *models* of real-life situations and how to use Javelin to create and work with models on a personal computer. If you have bought Javelin, you already want to do modeling on your PC. But even if you are just thinking about using Javelin, you will not be surprised to learn that you often model real-life situations on your PC: a simple checkbook spreadsheet is a model of your financial situation with your bank; a database of your address book is a (simplified) model of your social and business relations with other people; and a budget for your business or home is a model of your anticipated income and expenses.

A model is a representation of the real world in schematic form, which is constructed with a specific purpose in mind and omits details not relevant to that purpose. In the examples above, the checkbook does not reflect the value of your investments, nor your expected expenses or income. Similarly, although the address database lists your friends' names and addresses, it does not tell where you got to know them or how they are related to you. In each case, however, the details omitted are not relevant to the model. The checkbook model is perfectly adequate to tell you whether you can write a check for the electric bill right now; the address database is perfectly adequate to tell you what address to put on a letter to Aunt Sally (assuming you remember that her full name is Sally Green).

Differences Between Javelin and Lotus 1-2-3

■ You may wonder how Javelin differs from a spreadsheet program such as Lotus 1-2-3. Like Javelin, 1-2-3 is most frequently used to create models of the real world and ask questions of those models. However, the process of creating a model with Javelin is very different from the process you go through in 1-2-3.

Typically, to create a 1-2-3 spreadsheet, you envision the report you want to generate or the series of numbers you want to contemplate (for example, gross income versus profits), and you lay it out in 1-2-3's rows and columns. The relationships between the data in the various rows and columns evolve as you develop formulas that calculate your answers. The relationship between a cell and any other cell is defined by the formulas in those two cells. Nowhere is there

an overview of how everything fits together. In other words, the structure of your model is *implicit* rather than *explicit*.

This is helpful because you can develop models "on the fly" and work backwards from the result you have in mind, adding refinements as you go. It can create problems, however, when you go back to your spreadsheet and ask general questions about how the model works, such as How does the cost of printing affect quarterly expenses?

In Javelin, the relationships between your variables are explicit; Javelin can easily show you a list or draw a diagram of how monthly order rate and quarterly profit are related. However, instead of thinking visually, by putting a formula in the quarterly profit column to calculate the answer based on numbers in other columns, you can start thinking more formally, in terms of formulas: Total Expense = Printing Expense + Distribution Expense + Overhead Expense. Javelin includes a feature called the *Worksheet view* that allows you to think visually as you design your model, but the underlying structure of the model is still formula based.

The need for this type of formal thinking explains why we will spend some time describing the parts of a model and the thought processes behind the model—the time you spend thinking about the model, before you ever touch finger to keyboard, will be amply repaid later when you use or modify your model.

The Parts of a Model

■ In this chapter, you will invent and design a simple model that shows you the power of Javelin. But first let's examine the process of modeling so you can plan how your model should be put together.

Models have three parts: *inputs, intermediate calculations,* and *results.* (Javelin recognizes this and provides *views* to look at only inputs or only results.) Inputs represent the real-world information that you put into the model. Depending on the level of detail, they might signify units sold or the general level of economic activity. They are the data treated as "givens" in the model.

Intermediate calculations use the input data to create other data. If unit sales and unit price were inputs, they could be used to create gross sales, adjusted for collections to create cash receipts, and combined with expense

inputs to create cash on hand. The amount and complexity of calculations depend entirely on the nature of the model: some models have simple, straightforward calculations; others are lengthy and complex.

Results are the calculated values that tell you what you want to know. For a budget model, the results might be cash on hand at the end of the year; for a sales model, the results might be units sold in each period. Printed reports and graphs usually show the result values. The distinction between results and other calculated values is not a firm one, since intermediate calculations may also be interesting. For example, line items in a budget may be as important to know as the bottom line.

A good way to build a model is to consider the input data and desired results, and then build up the calculations needed to produce those results from the inputs. As you work on your model, you may well find that there are new results you need, which may require new inputs and calculations.

How to Design a Model

■ Before creating a model in Javelin, it is important to design it—that is, to decide what information to keep track of, how the data in the model will relate to each other, and what the key formulas will be. In designing and creating a model, you go through several stages: the *idea stage,* the *design stage,* the *implementation stage,* and the *reporting stage.* In the idea stage, you figure out what questions you want to answer and what information you will probably need to know. In the design stage, you think about how the information you have relates to the answers you want and what calculations will be needed to produce the results. In the implementation stage, you enter the information and relations into Javelin, creating the computer model. Finally, in the reporting stage, you produce reports that tell about your original idea.

In this chapter, you will complete the first two stages in the creation of a simple model. In Chapter 3, "Making a Model," you will complete the third and fourth stages, implementation and reporting using Javelin.

■ The Idea of the Model

The purpose of the sample model is to help set up a new company to sell books and records through the mail. The question is, Is this a good idea?

Unfortunately, no computer model will ever be able to answer this question, but such a model can tell what will happen *if you make certain assumptions.* Here are some assumptions that underlie this idea and therefore the model you are going to make for the idea.

The company's goal is to sell books and records profitably: this defines the goal of your model. With profit as the goal, the question Is this a good idea? becomes Given a set of assumptions, will the company make a profit? Profit doesn't have to be the goal of a model, but the goal must be expressible in measurable terms. For example, a philanthropic institution might have a goal of distributing as many books as possible without spending more than $500,000, which could be modeled too. However, if your goal is to create an interesting pastime, computer modeling cannot tell you whether distributing books and records will help you achieve it.

Once you have made this major assumption, several subordinate ones follow. How much money you make depends on how many books and records you sell, how much the books and records cost, and how much you charge for them. For each book or record that you sell, you have to spend some money to acquire it, advertise it, ship it, and so on. Other costs do not depend directly on how many books and records you sell—the telephone, the post office box, the rent—and you incur them regardless of how many (or how few) books and records you sell. These costs are known as *overhead.*

From these assumptions, you now know what information you need to answer the question. To determine if you'll make a profit, you have to know how many books and records you can sell, how much you have to spend on each book or record that you sell, how much to charge for each book or record, and what your overhead is.

■ *What Is a Variable?*

All data in a Javelin model (such as book sales, record sales, and profit) are stored as *variables,* which have names. A variable can be defined as a number or series of numbers (such as the number of books sold each month) or it can be defined by a formula calculated from other variables (such as total income from selling books and records).

There are two kinds of variables, depending on whether the value of the variable changes over time. Variables that always have the same value are called *constants.* Examples of constants might be Sales Tax Rate, which rarely changes, or Store Floor Area in Square Feet.

Variables that have different values over time are called *time-series variables*. Values may differ for each day, week, month, quarter, or year. For example, if you pay rent on the office every month, Rent is a monthly time-series variable. Bank Balance is a daily time-series variable because the amount of money in your account may change every day. Javelin allows you to have a mix of daily, weekly, monthly, quarterly, and yearly time-series variables as well as constants in your models—you must decide what period is appropriate for each variable you define. The time period associated with a variable is called its *periodicity*.

A variable can contain one of three types of information: numeric information, text information (also called *strings*), and date information. Most variables you use will contain numeric information, and for the next few chapters we will concentrate on numeric variables. Later we will demonstrate uses for storing other types of data as well. The ability to store and manipulate text and dates is new in Javelin PLUS: if you have Javelin 1.0 or 1.1, you can use only numeric information. Your Javelin model can contain up to 16,000 variables (the limit was 1,000 in Javelin 1.0).

For your model, you want an analysis of the book and record business on a monthly basis, so most variables will be monthly time-series variables, along with a few constants.

■ How to Name Variables

You can name variables whatever you want, but your model will be easier to build and understand if you give a little thought to the variable names you use. For example, in the book and record sales model, you have two categories of information: Sales and Expenses, and two products: Books and Records. In this case, reasonable variable names would be Book Sales, Book Expenses, Record Sales, and Record Expenses. If later on you decide to further categorize your data by geographical region (say, Eastern and Western), useful names would be Eastern Book Sales, Western Record Expenses, and so forth.

Variable names are important because when you set up reports, the names give structure to the reports. For example, with the variable names just chosen, it would be easy to build a sales report categorized by product, or to build product reports containing sales and expense numbers. In Chapter 7, "Defining Variables and Entering Data Using Worksheets," you will see how to build reports like this.

Computers are notoriously literal-minded, so consistency will give you the best use of variable names. For example, if your sales variables are

named Book Sales and Record Sales, the corresponding expense variables should be named Book Expenses and Record Expenses rather than Cost of Books or Recording Expenses. If your names are not consistent, your model will still work but building reports will be considerably more difficult.

In short, variable names should categorize your data in ways useful to you. If you select variables for regions, the names should contain the region names; likewise, variable names for products should contain the names of the products.

■ How to Design a Model

How does this information about numbers of books and records sold, cost per book or record, and overhead produce a profit figure? You start from the assumption that

Profit = Total Sales – Total Expenses

Not only have you defined the nature of profit in your model, you have defined three variables as well. Now you are interested in total sales and total expenses. You sell only books and records, so

Total Sales = Book Sales + Record Sales

To start, assume that all books sell for one price and all records sell for another fixed price. Later you can change this assumption without having to completely reconstruct the model. For the time being, however,

Book Sales = Book Volume * Book Price

and

Record Sales = Record Volume * Record Price

The asterisk represents multiplication. Volume represents the number of books or records sold in each month. (The figures given for costs, percentages, and other parts of these sample models are for example only and may not reflect actual numbers in the real world.)

This covers the Total Sales part of Profit. The Total Expenses side looks similar:

Total Expenses = Overhead + Book Expenses + Record Expenses

We mentioned Overhead earlier: money that must be spent regardless of how much you sell. In this model, assume that your overhead costs are the same every month, which means that Overhead is a constant.

Because you assume that each book and each record sells for a fixed price, Book Price and Record Price are also constants. For the time being, assume that each book and record costs you some percentage of its sales price, say 75 percent for books and 50 percent for records. Your book and record expenses then are

Book Expenses = Book Volume * (0.75 * Book Price)

and

Record Expenses = Record Volume * (0.50 * Record Price)

That is, for each book sold, 75 percent of its price is an expense for you, and for each record sold, 50 percent of its price is an expense.

With this, you have defined your model. You will certainly make it more complex later on, but for now, it is a simple set of assumptions and relations that will give you a simple answer.

■ *How to Make Assumptions*

In the idea stage of your model, you made certain general assumptions, such as the goal of the model (profit) and the general relationship between the parts of the model (profit depends on books and records sold). In the design stage, you made more specific assumptions, creating variables and defining their relationships, such as Total Expenses equaling Overhead plus Book Expenses plus Record Expenses. As you implement the model, you will make further assumptions by entering specific values for the variables.

The early assumptions that affect the general structure of the model are the hardest to change later on, because changing them implies changing many formulas and reorganizing a large part of the model. Clearly you have to make some assumptions at each stage of your model or it will never get built. Putting some thought into the early model design may save more trouble later.

The easiest assumptions to change are the specific values of the variables, so assign values to the constants now. You'll assign values to the time-series variables later when you actually create the model in Javelin.

For example, you may decide that your Overhead will be $500 a month, your Book Price is $7.50, and your Record Price is $12.50. How do you enter these assumptions in Javelin? Assuming specific values for constants in Javelin is done with simple formulas:

```
Overhead = 500
Book Price = 7.50
Record Price = 12.50
```

Summary: The Model's Variables and Formulas

To summarize the model, Figure 1.1 lists the variables with their formulas. The input variables are Book Volume, Book Price, Record Volume, Record Price, and Overhead, because they are not defined by formulas. Book Price, Record Price, and Overhead are constants, with one constant value for each variable. Book Volume and Record Volume are monthly time-series variables, and you will enter a value for each month for each of them.

The intermediate calculations are Book Sales, Record Sales, Book Expenses, Record Expenses, Total Sales, and Total Expenses. These are all monthly time-series variables that you calculate to get your result.

The result variable is Profit, another monthly time-series variable, which answers the question posed by this model, Will the business make a profit?

Good Modeling Practices

That is all there is to designing a complete, useful Javelin model. Beyond following the logic of what you did, there are several important lessons to learn from this process.

Plan Ahead

The first lesson is the importance of planning ahead. Once you decided to model your book and record business, you found you had to make some assumptions, the most important of which was deciding how to express the goal of your model in measurable terms. While any goal that can be

```
Profit = Total Sales - Total Expenses
. Total Sales = Book Sales + Record Sales
. . Book Sales = Book Volume * Book Price
. . . Book Price = 7.5
. . Record Sales = Record Volume * Record Price
. . . Record Price = 12.5
. Total Expenses = Overhead + Book Expenses + Record Expenses
. . Overhead = 500
. . Book Expenses = Book Volume * (0.75 * Book Price)
. . Record Expenses = Record Volume * (0.5 * Record Price)
```

Figure 1.1: Variables for Books and Records model

expressed numerically will do, you decided that your goal would be profits. You then decided that the basic elements of your business were money spent (expenses) and money earned (sales), and that you had two kinds of expenses: those related to selling an individual book or record and general overhead required to stay in business. While these are all obvious assumptions for this model, they make a very important point: think carefully about your model before you start building it.

■ *Design from the Top Down*

The second lesson in the model-designing process is the importance of *top-down design*. In other words, start by thinking about the most general parts of your model and only later move on to specific details. Take another look at your formulas:

Profit = Total Sales – Total Expenses
Total Sales = Book Sales + Record Sales
Book Sales = Book Volume * Book Price
Record Sales = Record Volume * Record Price

You could write your Profit formula as

Profit = Book Sales + Record Sales – Total Expenses

or even as

Profit = Book Volume * Book Price + Record Volume * Record Price – Total Expenses

However, if you do this and later decide to add another product line, you have to change the Profit formula. Now, the Profit formula is not only hard to read and understand, but any future changes will make it even more complicated. For instance, imagine what would happen if you decided to sell books for different prices. But by calculating a variable called Book Sales, no matter how complicated your book prices become, you never have to change the Total Sales formula. Similarly, if you calculate a number called Total Sales, you never have to change the Profit formula. Put another way, this means that every time you make a change to the model, you change only those variables directly concerned with what you are doing and avoid changing a formula that deals with Record Volume when you are really concerned with Book Price.

This is an example of top-down thinking, which works out the answers to general questions (Total Sales) and leaves the specific questions (Book Sales) for later. It is very convenient if you are the type of person who thinks that way, but for those of us who are not, top-down thinking requires some discipline. Models can grow in an unplanned way, but to make them easy to use and change requires a certain amount of structure, and the best way to achieve it is to start at the top.

The way you developed your constant values (Overhead, Book Price, and Record Price) also shows the advantages of top-down thinking. You said

```
Book Sales = Book Volume * Book Price
Book Price = 7.50
```

when you could have said

```
Book Sales = Book Volume * 7.50
```

There are two disadvantages to this latter method. First, there is no identification of what the number 7.50 stands for. Second, if you decide to change the value in the model ($7.50 book price), you have to make a change to the logic of the model, that is, to a formula. This technique leads to errors. Third, you might decide later that book price should be a time-series variable rather than a constant. You can change the single variable Book Price without changing any other variables.

Top-down considerations make even more difference when your models grow larger and more complex.

■ Make Models the Right Size

Most people start out by building one huge model to represent everything they ever deal with. That is practically the only way to build a model with a spreadsheet program, but in Javelin it is usually a bad idea.

The larger your model is, the harder it is to use and to keep mistakes out. Large models calculate more slowly than smaller ones, and they can become too large to fit on a single diskette. Javelin provides several facilities to link models together, so that changes in one model are automatically reflected in another. Using these facilities lets you build several small linked models rather than one huge one, which can be a big help in managing your information.

Small linked models can also be helpful when several people are working on a modeling job. Keeping individual models separate allows people to work in concert but to avoid the confusion that ensues when several people try to change one model at the same time.

There is no simple formula for determining the right size for a model, but if you find yourself looking at part of a model and saying, "I don't remember putting in that part—I wonder what it does?," you should try breaking up your model into several smaller ones.

■ Base Models on Decision Items

Try to make the inputs to your model the actual items that people make decisions on, rather than quantities derived from them. The model can calculate the other quantities for you. For example, if you are creating a budget, rather than making Travel Budget an input, make Number of Trips Taken the input, and calculate the travel budget by multiplying it by an average trip cost. For one thing, people can probably estimate the number of trips they plan to take far more accurately than they can estimate how much the trips will cost. For another, if you ask several managers how much to budget for 15 trips around the country, you will get different answers—asking them about the decision item should get you more consistent numbers.

■ Model Flows Rather than Stocks

Your models will usually have some variables that represent *flows* (e.g., Monthly Cash Received or Quarterly New Hires) and others that represent

stocks (e.g., Monthly Cash Balances or Number of Employees). You should generally make the flow the input variable and calculate the related stock from it. Managers know their flows better than their stocks; for example, how many people they plan to hire rather than how many people they will have working for them, or how much income they expect rather than how much cash will end up in the bank.

■ Choose Appropriate Time Periods

Javelin offers you a wide variety of time periods that you can use to set up your data. Data can be daily, weekly, monthly, quarterly, yearly, or it can be constant (unaffected by time). Using the right time periods for your variables will make your model easier to use and to modify.

If your data are time related, define them to Javelin that way. If you are making a tax model for 1987, say, you could make everything a constant, since the model is only for one year. Don't do it. If an item is for 1987, define it as a yearly variable. By doing so, you make it possible to use Javelin's powerful date-conversion facilities, you let Javelin do better error checking, and when the next year comes around, you can adapt the model easily.

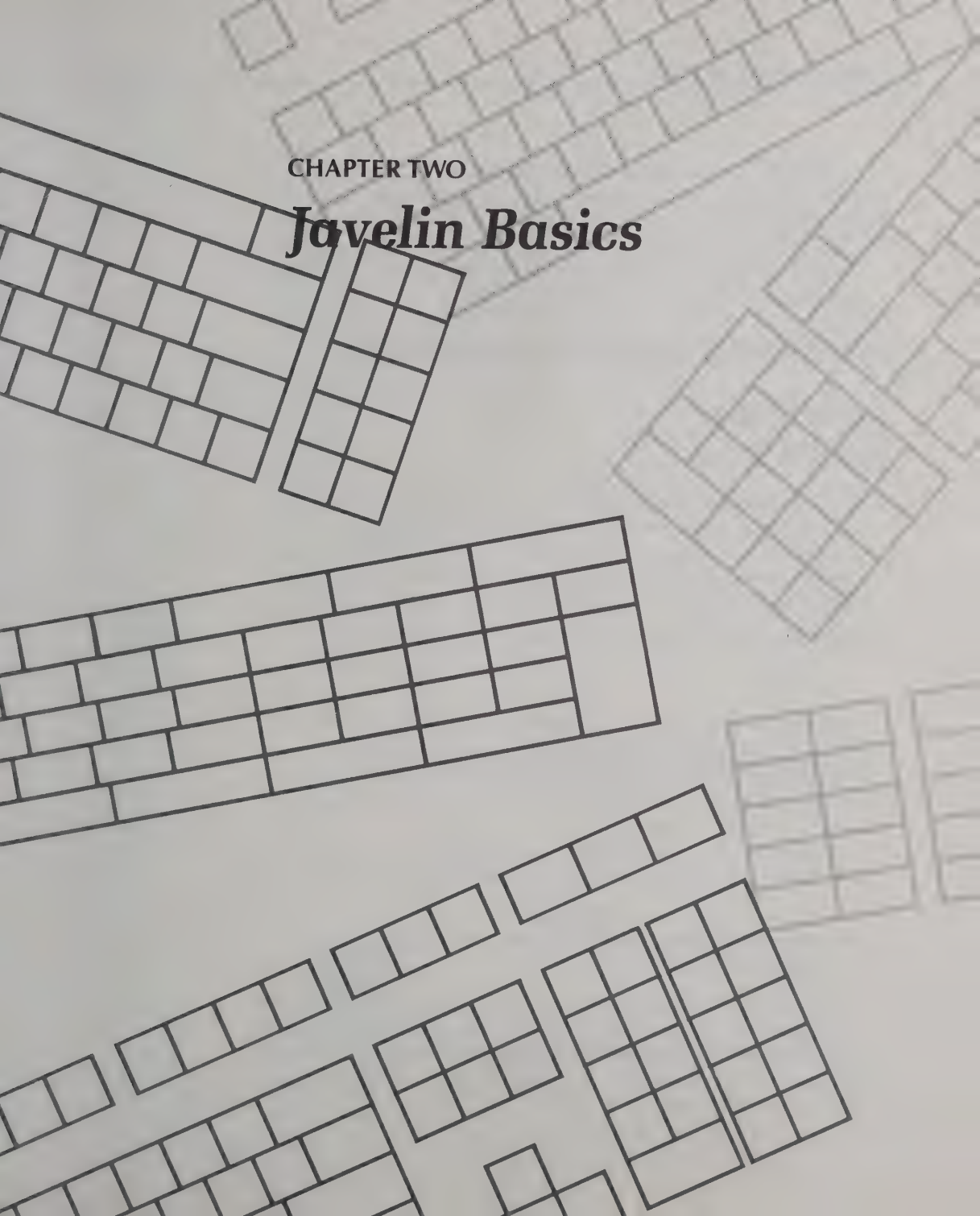
For your model, you might decide to calculate all time-series variables on a monthly basis because you place your book and record orders once a month or because you want monthly results. A model can contain variables with different periods. Your sales and expense variables could be monthly, but you could define profit to be quarterly. Javelin will do the necessary conversions (see Chapter 9, "Javelin and Time").

Conclusion

■ Before you can implement this model in Javelin, you must know how to make Javelin do what you want—how to give commands and enter formulas and data. In Chapter 2, "Javelin Basics," we will review the basic principles of using Javelin, so that in Chapter 3, "Making a Model," you can create this model.

CHAPTER TWO

Javelin Basics



So far, we have talked about what makes a good model and how to design one. Before you can start using Javelin, you should know how to make it do what you want—how to issue commands, how to enter formulas and data, what's on the screen, and what the keys do. We'll cover these topics in this chapter, so that in Chapter 3, "Making a Model," you will be ready to implement the model you designed in Chapter 1.

Parts of the Javelin Screen

■ The Javelin screen displays information about the model you are working on, as well as status information about the particular command you are giving or information you are entering. Once you have followed the installation instructions that come with the program and are at the DOS prompt, start up Javelin to see what the parts of the screen look like and to try out some keys. Type

```
JAV
```

and press the Enter key (⟵). Once Javelin has finished loading, the Javelin 1.0 and 1.1 screen looks like Figure 2.1.

If you have Javelin PLUS, your screen looks like Figure 2.2. On the second line of your screen you'll see the Javelin serial number on the left, and the version of Javelin on the right.

The top two lines are the *edit line* and the *information line,* the middle area of the screen is the *view area,* and the bottom line is the *status line.*

■ The Edit Line

When you enter information, such as data or formulas, the characters you type appear on the edit line. When you press ⟵, the information is stored in your model and disappears from the edit line. Prompts from Javelin also appear on the edit line when the program asks you for the name of a variable or other information.

■ The Information Line

The information line is the second line on the screen, where Javelin displays information about what you are doing. Messages appear as you enter information and give commands, prompting you for what to do next.

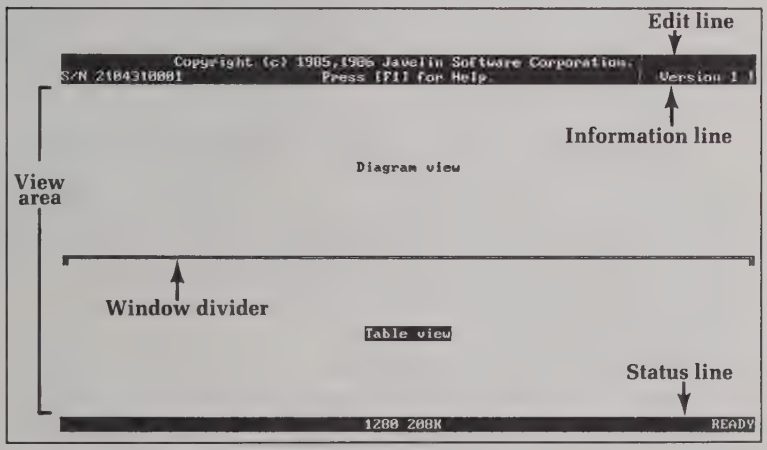


Figure 2.1: The Javelin copyright screen

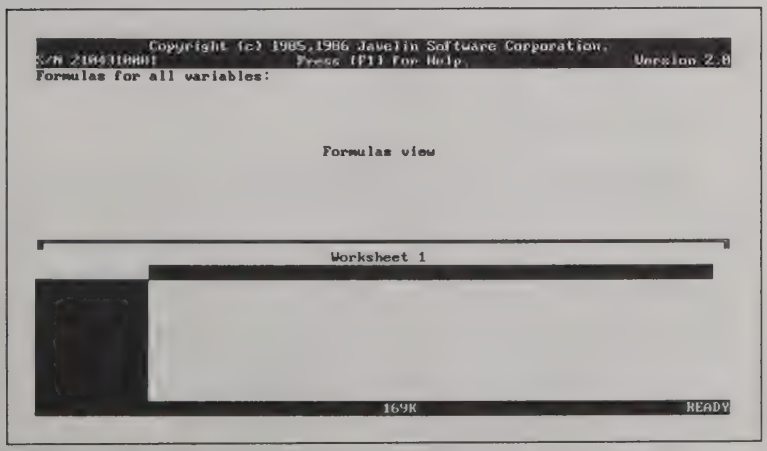


Figure 2.2: The Javelin PLUS copyright screen

■ The Status Line

The bottom line of the screen is the status line, which displays a number of different *status indicators,* one-word messages that tell you about the model you are working on.

On the far left of the status line, the name of the model appears. If you have changed or added to the model since the last time you saved it to a disk, Javelin displays an asterisk after the model's name. If you haven't saved it yet, no name is displayed (a model doesn't have a name until it is saved on a disk or named with the / **D**efine **S**ettings command).

In the middle of the status line, the current amount of free memory is noted. All data, formulas, and settings in a model are kept in the PC's memory while you are working. Therefore, the size of your model is limited to what Javelin can fit into memory. Memory is measured in *K,* thousands of bytes. A *byte* is enough space to hold one character of text. Your PC must have a total of at least 512K to run Javelin. If you have an expanded memory board (such as the Intel AboveBoard or the AST RAMpage or Premier), you see two numbers, one for regular memory and one for expanded memory.

At the right end of the status line is an indicator that tells you the general state of the program, or the *mode.* For example, when the READY status indicator is displayed, Javelin is in Ready mode. Other indicators may appear in other locations on the status line, such as the CAPS indicator to remind you that all letters you type will be capitalized. The indicators you may see are listed in Table 2.1. Now Javelin is in the Ready mode.

■ The View Area

The area between the information line and the status line is called the view area. This is where you see different displays or *views* of your model. One view of the model can take up the entire view area, or you can divide the area into two *windows* with one view in each. When Javelin starts, the screen is divided into two windows. There are commands to display one or two windows (see Chapter 6, "Javelin Views and Windows," for more on the view area, windows, and views you can display). Javelin 1.1 starts with a Diagram view and a Table view on the screen. Javelin PLUS starts with a Formulas view and a Worksheet view.

The two windows are divided by a double line, called the *window divider,* with corners on each end. The cursor appears in the window in which you are working, called the *active* window. This is the view Javelin will change, based on what you type on the edit line. When the view in the top

MODE INDICATORS

Indicator	Meaning
ERROR	There is an error in the model. See Chapter 11, "Special Values and Diagnosing Problems," for how to find and correct it.
MENU	You are looking at a menu or a settings sheet.
POINT	You have pressed the Point key, F4 (see Chapter 8, "Formulas and Functions").
READY	Javelin is waiting for the next command or entry—it is not in the middle of an operation. When the READY indicator is displayed, Javelin is in Ready mode.
WAIT	Javelin is doing something that takes time—at least a few seconds. You can type while WAIT is flashing—Javelin remembers what you type and accepts the keystrokes when the operation is over.

OTHER INDICATORS

Indicator	Meaning
CALC	Recalculate the model before printing it, because data or formulas have been changed since the model was last recalculated (see Table 2.2).
CAPS	Caps Lock is on, so that all the letters you type are capitalized. To turn Caps Lock on or off, press the Caps Lock key. Javelin does not distinguish between capital and lowercase letters.
NUM	Numeric Lock is on, so that the keys on the numeric keypad enter numbers rather than moving the cursor. To turn Numeric Lock on or off, press the Num Lock key.
PAUSE	You are running a macro that has paused for you to enter information. To continue running the macro, press the F7 key (see Chapter 14, "Macros").
RECORD	You are recording a macro. See Chapter 14 for how to do this.
STEP	You are testing a macro in single-step mode. See Chapter 14 for how to do this. To stop running the macro, press Esc.
SYNC	The two windows are synchronized. Synchronization is turned on and off by pressing F3, the Sync key (see Table 2.2).

Table 2.1: Indicators on the status line

window is active, the corners of the window divider point up, and when the bottom window is active, they point down.

To switch the window you are working on, press the Window key, F6. You notice that the corners on the window divider switch directions, indicating which window is active. There is also a highlight in the active window: what is highlighted depends on what view is displayed in the window.

■ The Cursor

The *cursor* indicates where you are working at the moment. It is analogous to the pencil point when you write on paper. The form of the cursor depends on what view you are in. It is usually a highlighted box around the item you are working on. When you type in information, a blinking underline called the *cue* appears on the edit line, showing where the information you type will go.

The PC Keyboard and Javelin

■ Most of the keys on the keyboard work the same in any program—the letter keys, the number keys on the top line of the keyboard, and punctuation such as the period and the comma. But other keys work differently in different programs. Many keys and commands work the same way in Javelin as in Lotus 1-2-3. For example, most of the function-key meanings are the same, and you press the / key to see the menu. So when in doubt, try the 1-2-3 method and it will probably work.

Even if you are comfortable using the PC keyboard with 1-2-3 or other software, take a moment to review what each key does in Javelin.

Depending on what computer you have, the layout of your keyboard may be different from the one described here, which is the standard IBM PC keyboard (see Figure 2.3). The keyboard can be divided into three general areas: the *function keys,* the *typewriter keys,* and the *numeric keypad.*

■ The Function Keys

Function keys perform special tasks that are repeated frequently. Rather than typing in a Javelin command, you can perform it by pressing just one function key. The meaning of each key is shown in Table 2.2. For the more advanced function keys, we will refer you to the chapters that explain their

use more fully. You should have received a template with Javelin that shows the function key names and fits around the keys.

The Typewriter Keys

The main section of the keyboard contains mostly letters, numbers, or punctuation characters, just like a typewriter. However, some keys in this part of the keyboard are unique to computers, and their special meanings in Javelin are shown in Table 2.3.

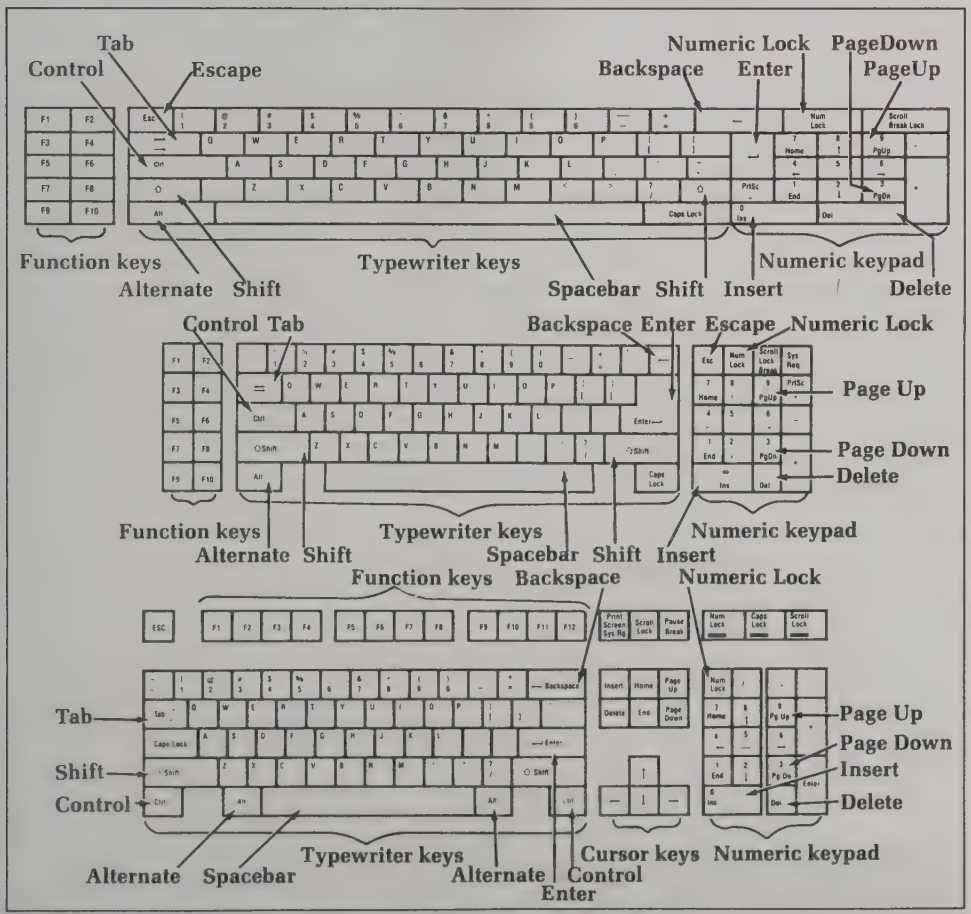


Figure 2.3: The PC/XT, PC/AT, and Enhanced PC keyboards

Key Name on Keyboard	Name in Javelin	Description
F1	Help	Shows help screens that contain information about Javelin.
F2	Edit	Lets you edit information you have already entered.
F3	Sync	Tells Javelin to keep two screen windows synchronized with each other. As you move around one window, corresponding information is displayed in the other (see Chapter 6, "Javelin Views and Windows").
Alt-F3	Sticky Sync	Works like Sync (F3) but stays on even if you switch windows with F6 (see Chapter 6). Javelin PLUS only.
F4	Point	Lets you point to a variable name on screen rather than typing it in (see Chapter 8, "Formulas and Functions").
F5	Goto	Brings a particular variable or building block into view on the screen (see Chapter 5, "Special Keys and Settings Sheets").
F6	Window	Moves the cursor from one window to the other when two windows are on screen. Does not function with only one window on screen.
F7	Macro	Runs a macro (see Chapter 14, "Macros").
Alt-F7	Single-Step Macro	Runs a macro one step at a time (see Chapter 14).
F8	Quick-graph	Displays a Quick-graph of the variable you are looking at (see Chapter 6).
F9	Calc	Recalculates all the formulas in the model, based on the latest values of the data.
F10	Graph	Displays the graph most recently displayed or worked on (see Chapter 12, "Graphing").

Table 2.2: The function keys

Key Name on Keyboard	Name in Javelin	Description
Esc	Escape	Cancels a command.
→|	Spell	Labeled as Tab or →|. When you type a name, such as a variable name or file name, the Spell key shows a list of possible responses. If only one response is possible, Javelin spells it out.
Ctrl	Control	Used like a special shift key in combination with other keys. Does nothing by itself.
Shift	Shift	Used to capitalize typed letters.
Alt	Alternate	Used for running single-letter-name macros (see Chapter 14, "Macros").
/	Slash	Displays the Javelin command menu.
←	Backspace	Erases the character you just typed.
↵	Enter	When you are typing an entry, tells Javelin to store the entry. When you are looking at a command menu, tells Javelin to execute the command the cursor is on.
PrtSc	Print Screen	Pressed by itself, types an asterisk. If you hold down Shift and press PrtSc, Javelin prints the contents of the screen on your printer. If your printer is not set up to print graphics characters, lines and other special characters may be printed as letters.
Caps Lock	Caps Lock	Capitalizes typed letters until released by pressing a second time. Unlike a typewriter, it affects only the letter keys. For example, it does not make the number keys on the top row of the keyboard print punctuation. When Caps Lock is on, the word CAPS appears on the status line.
Num Lock	Numeric Lock	Turns Numeric Lock mode on and off. See Table 2.4 for keys affected by Numeric Lock.

Table 2.3: Special keys

The Numeric Keypad

The keys on the numeric keypad can be used in two ways—to move the cursor or to type numbers. The meanings of these keys depend on whether Numeric Lock mode is on or off. When Numeric Lock mode is on, the keys take on number-entering meanings. When it is off, they have cursor-movement meanings.

To turn Numeric Lock mode on and off, use the Num Lock key. When the PC starts up, Numeric Lock mode is off. Every time you press Num Lock, you flip Numeric Lock the other way. The indicator NUM appears on the status line when Numeric Lock mode is on. Table 2.4 shows the meanings of

KEYS AFFECTED BY NUMERIC LOCK

Key Name on Keyboard	Description
←	Moves the cursor to the left.
→	Moves the cursor to the right.
↑	Moves the cursor one line up.
↓	Moves the cursor one line down.
Home	Moves the cursor to the beginning of whatever you are looking at.
End	Moves the cursor to the last item in whatever you are looking at.
PgUp	Displays the previous page of information.
PgDn	Displays the next page of information.
Ins	In a worksheet, inserts a row. In a table, chart, or quick graph inserts space for a new value.
Del	Deletes one character or one item you are working on.

KEYS UNAFFECTED BY NUMERIC LOCK

Key Name on Keyboard	Description
Scroll Lock	Not used in Javelin.
Gray Plus	Used to type the addition symbol.
Gray Minus	Used to type the subtraction symbol.

Table 2.4: The numeric keypad when Numeric Lock mode is off

these keys when Numeric Lock mode is off and they have cursor-movement meanings.

Two Basic Function Keys

■ Two function keys you need immediately are the Help and Calc keys. The Help key, F1, displays helpful information on the screen, and the Calc key, F9, recalculates all formulas in the model based on the latest information you have entered.

■ The Help Key

Javelin has a handy feature called *on-line help*. Whenever you wonder how to use a key or a command, or have modeled yourself into a corner, pressing F1 calls up information about using Javelin. The information is displayed one screen at a time; each screenful of information is called a *help screen*.

Javelin tries to guess what you want information about, based on what you are doing. Usually it guesses well. However, if it does not, you can change help screens until you reach one that contains the information you need: highlighted phrases on the screen guide you to other help screens. Moving the cursor to a phrase and pressing ⟵ brings help about that phrase onto the screen.

When you are done looking at help screens, press Esc to return to Javelin at exactly the place where you first asked for help.

Ask for help by pressing F1, which produces the help screen appropriate for your location. For example, if your screen shows a Table view in the active window, you'll see a help screen like Figure 2.4.

This information comes from the Javelin manual, but it's sometimes more convenient to call it up on the screen with one keystroke. Your cursor is on the bottom line of the screen. Move the cursor to the help topic Help Index and ask to see it. To accomplish this, press the right-arrow key (→) twice, then press ⟵.

The help index (see Figure 2.5) shows a menu of the major help topics and allows you to browse through help screens until you find what you need.

When you are done looking at the help screens, press Esc to return to the exact point in Javelin where you left off.

The Calc Key

As you add and edit formulas and data, Javelin stores this information in your model, but it doesn't calculate everything right away. Instead, the CALC

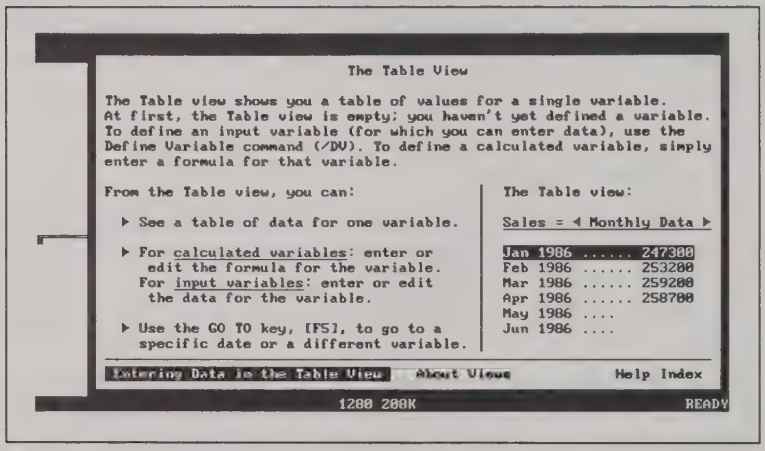

The Table View
The Table view shows you a table of values for a single variable.
At first, the Table view is empty; you haven't yet defined a variable.
To define an input variable (for which you can enter data), use the
Define Variable command (/DV). To define a calculated variable, simply
enter a formula for that variable.
From the Table view, you can:
▶ See a table of data for one variable.
▶ For calculated variables: enter or edit the formula for the variable. For input variables: enter or edit the data for the variable.
▶ Use the GO TO key, [F5], to go to a specific date or a different variable.
The Table view:
Sales = ◀ Monthly Data ▶
Jan 1986 247300
Feb 1986 253200
Mar 1986 259200
Apr 1986 258700
May 1986 ....
Jun 1986 ....
Entering Data in the Table View
About Views
Help Index
1200 208K
READY

Figure 2.4: Help screen for Table view

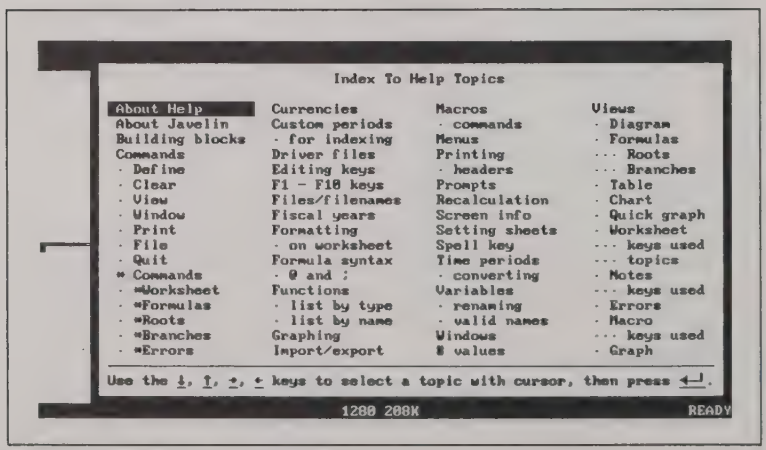


Figure 2.5: The help index in Javelin 1.1

indicator appears in the lower-right corner of the screen. You have to press the Calc key, F9, to recalculate all the variables.

Unlike many spreadsheet programs, Javelin does no automatic recalculation. As you may have noticed in developing large spreadsheets, recalculation can take quite a while, and that long pause after every change can be distracting. Therefore, most spreadsheets allow you to turn off automatic recalculation, and most people do. In Javelin, automatic recalculation is always off.

If you have made changes to the model since it was last recalculated, the CALC indicator appears on the status line. Be sure to recalculate the model before you print reports to keep the information up-to-date.

Javelin uses the Calc key to calculate new values for all variables and information in the entire model. During calculation, the WAIT indicator blinks in the lower-right corner of the screen on the status line. After a short pause, the numbers are updated (if you haven't entered anything new into your model, nothing happens). A large, complex model may take many seconds, even minutes, to recalculate.

In Javelin 1.1 and PLUS, as it recalculates Javelin displays on the status line the number of formulas still to be recalculated; the number decreases to give you an idea of how much recalculation remains.

Javelin Commands

■ When you use Javelin, you either add and edit information in your model or issue commands that tell Javelin how to display, store, format, print, graph, or otherwise manipulate the model. To give Javelin a command, you first display a *command menu.* You make choices from this and succeeding menus until you have told it exactly what you want. Javelin then performs the command. You must be in Ready mode to begin a command.

The first command menu that Javelin displays is called the *Main menu* (see Figure 2.6). To display the Main menu, press /.

When a menu is displayed, the status indicator changes from READY to MENU, signifying that you are in Menu mode. You stay in Menu mode until you have finished giving the command.

The Main Menu

In the Main menu, menu choices are displayed across the third line of the screen. Each word is a menu choice. One choice is highlighted, meaning that the menu cursor is on that choice. The information line (the second line of the screen) has a description of the menu choice that the cursor is on.

The first seven choices are always the same. Depending on what you are viewing, there may be an eighth choice, which always begins with an asterisk. The choices on this eighth menu perform functions that are meaningful only in the view you are in.

How to Choose a Command from a Menu

You may move the cursor to the other menu choices by using the arrow keys, ← and →. Or you can move the cursor directly to a menu choice by pressing the first letter of that choice. In all Javelin menus, each choice begins with a different letter. Either of these movements is called *selecting* that menu choice. Press ← and → a few times to see all the choices on the Main menu.

Options Menus

Each choice on the Main menu has a number of options listed below it in an *options menu*. The menu of options rolls down below each menu choice when the cursor is on it, and each options menu has its own cursor. The option the cursor is on is highlighted. Figure 2.7 shows the menu of options for the / **D**efine command (that is, if you press / and select Define).

Again, to choose a command from an options menu, you can move the cursor using arrow keys, this time Up (↑) and Down (↓), or you can press the first letter of the option you want.

How to Back Out of a Command

After beginning to give a command, you may want to back out of it. Press Esc to back up one step, returning to the previous menu. You can keep pressing Esc until you are no longer in a menu, and the status indicator changes from MENU back to READY. To practice backing out of the menus,

press Esc to cancel the options menu, then press it again to cancel the Main menu.

Diagram views show how variables in the model are connected. If you are using Javelin 1.1, you already have a Diagram view in the upper window. Give the command / **V**iew **D**iagram to display a view of the model in the current

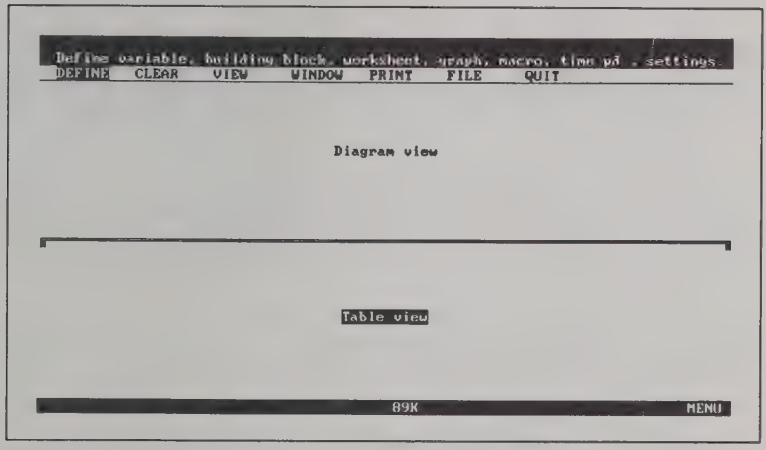

Define variable, building block, worksheet, graph, macro, time pd , settings.
DEFINE CLEAR VIEW WINDOW PRINT FILE QUIT
Diagram view
Table view
89K
MENU

Figure 2.6: The Main menu

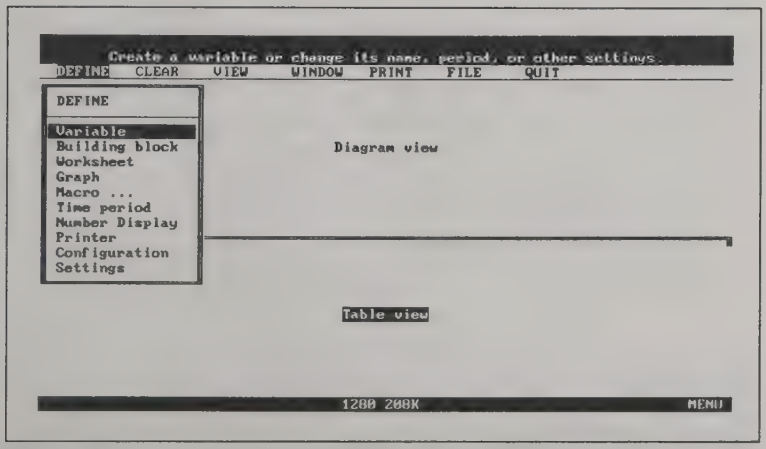


Figure 2.7: Main menu with options menu for / **D**efine command

(lower) window. That is, press / and select View by using ← and → or by pressing V. Notice the description of the View choice on the information line. Select Diagram by using ↑ and ↓ and pressing ⏎, or by pressing D.

Javelin now displays a Diagram view in the window you are working in. The status indicator changes from MENU back to READY.

■ More Options

Some commands have even more options. For example, the / **V**iew **F**ormulas command, which displays a list of the formulas in your model, has a further option in which you can indicate how you want the formulas arranged. Notice that the Formulas option has three dots after it. This means that there are further options to be chosen under Formulas, so select that. Another options menu appears, overlapping the View options menu (see Figure 2.8).

You have three choices: display all formulas, display just the formulas that feed into a particular variable (*roots*), or display just the formulas affected by a particular formula (*branches*). Select All. Javelin displays a Formulas view (if you are using Javelin PLUS, there is already a Formulas view in the upper window).

■ Memory in Menus

When you press / to see the Main menu, the cursor is on the last command you gave, which makes it easy to give the same command several times. Also, in each options menu, the cursor is on the last option chosen from that menu. Javelin remembers your choices and makes it easy to repeat them.

Look at the Main menu again by pressing /. Since the last command you gave was / **V**iew **F**ormulas **A**ll, the View option is highlighted. Cancel the menu by pressing Esc.

Table 2.5 lists and describes the seven commands that always appear on Javelin's Main menu.

How to Quit Javelin

■ When you are ready to stop working in Javelin, use the / **Q**uit command to exit from the program and return to DOS. Before leaving Javelin, you

must save the model you are working with on a disk or your work will be lost (see Chapter 4, "Managing Files and Printing Views," for file commands). Now give the command to quit Javelin.

The Quit menu appears and gives you the choice of quitting, canceling the / **Q**uit command and returning to Javelin, or saving the current model and then quitting (see Figure 2.9).

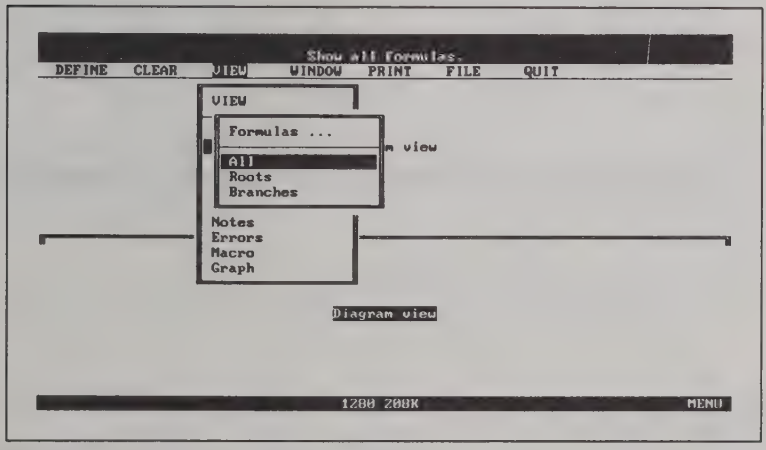


Figure 2.8: Menu with second level of options

Command	Description
DEFINE	Creates or modifies parts of the model.
CLEAR	Erases parts of the model.
VIEW	Displays one of the ten Javelin views of the model in the current window.
WINDOW	Controls the number and size of the windows on the screen (each window contains a view).
PRINT	Prints a view of the model.
FILE	Saves the model on the disk, loads a previously saved model, or executes other disk-related commands.
QUIT	Exits Javelin.

Table 2.5: Commands on Main menu

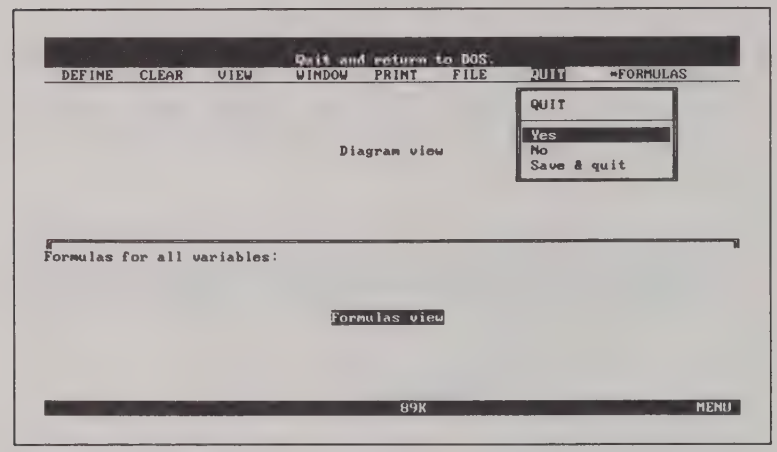
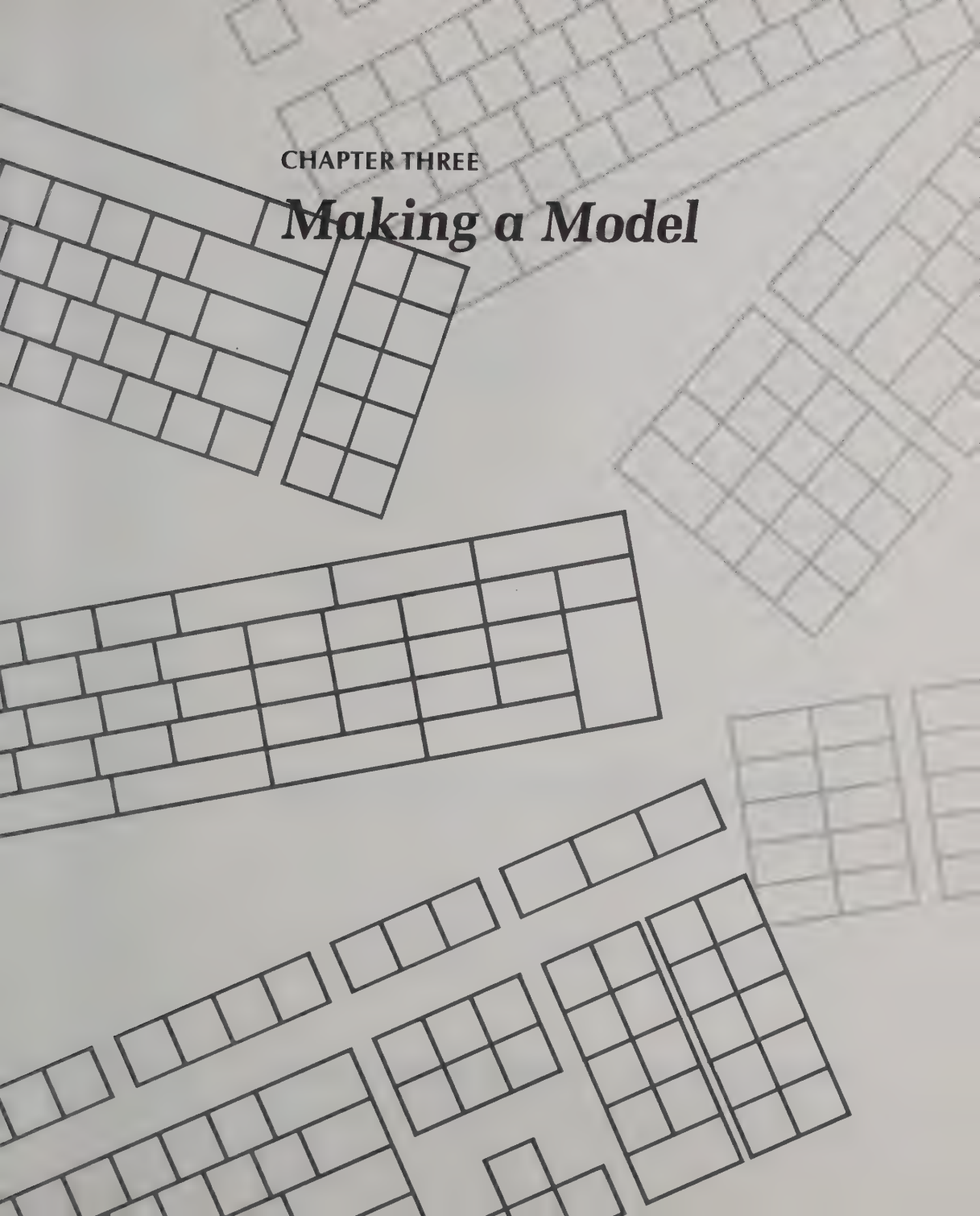


Figure 2.9: The Quit menu

Go ahead and select Yes to quit—you haven't done anything worth saving. Javelin stops running and you return to DOS.

Conclusion

■ You now have an overall idea of how to "drive" Javelin—you know the parts of the screen, how the keys work, and how to give a command. In the next chapter, you will use this information while creating the profit-and-loss model of book and record sales you designed in Chapter 1. Then you will learn more about the most common commands.



CHAPTER THREE

Making a Model

You are now ready to use Javelin to create the model you designed in Chapter 1. Implementing your model is almost as simple as typing in the relationships you defined (summarized in Figure 1.1). To type them in, you will use some of the commands and function keys described in Chapter 2. Enter Javelin and clear the copyright notice from the screen as you did in Chapter 2.

If you are using Javelin PLUS, the lower window shows a Worksheet view. To create this model, you'll use a Table view to enter formulas and data. With the lower window active, give the command / **V**iew **T**able ⟵.

If you are using Javelin 1.0 or 1.1, show the Formulas view in the upper window, so you can see the list of formulas grow. Press F6 and give the command / **V**iew **F**ormulas **A**ll.

Entering Formulas

- Start by entering the formulas for the variables in your model. Type

  Profit = Total Sales – Total Expenses

Then press ⟵. (We won't keep reminding you to press ⟵, since Javelin requires you to do so to end each command or item of keyboard input.) Your screen should now look like Figure 3.1.

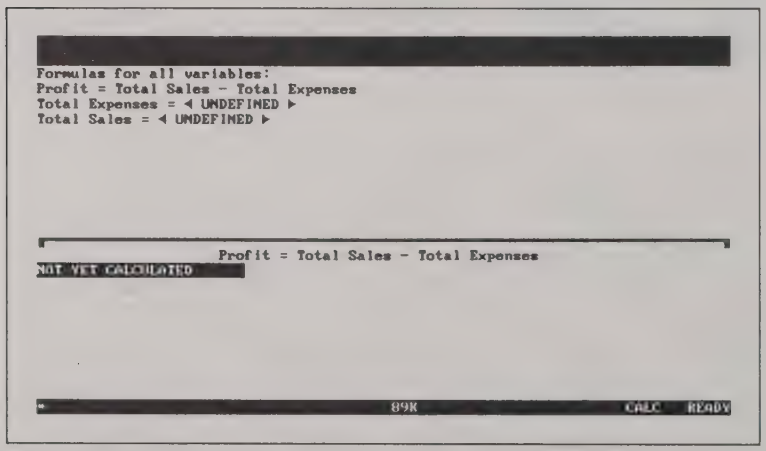

Formulas for all variables:
Profit = Total Sales - Total Expenses
Total Expenses = ◄ UNDEFINED ►
Total Sales = ◄ UNDEFINED ►
Profit = Total Sales - Total Expenses
NOT YET CALCULATED
89K
CALC
READY

Figure 3.1: Display of Profit formula

Notice that on both halves of the screen (top and bottom), the formula you typed is displayed. Javelin is showing you two different views of your model. We'll talk more about views in just a moment and discuss them in detail in Chapter 6, "Javelin Views and Windows." Now type

Total Sales = Book Sales + Record Sales

Your Total Sales formula appears on the bottom half of the screen, and is added to the list in the top view. To complete the model, type the following formulas (and press ⟵ after each):

Book Sales = Book Volume * Book Price
Record Sales = Record Volume * Record Price
Total Expenses = Overhead + Book Expenses + Record Expenses
Book Expenses = Book Volume * (0.75 * Book Price)
Record Expenses = Record Volume * (0.50 * Record Price)

The final elements of your model are the values you defined for Overhead, Book Price, and Record Price. Those constant values are entered into Javelin as simple formulas. Type

Overhead = 500

Your screen should look like Figure 3.2.

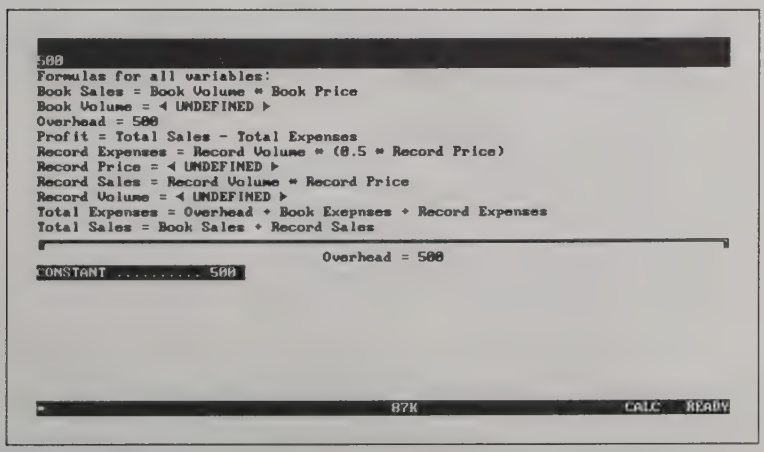

500
Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◀ UNDEFINED ▶
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = ◀ UNDEFINED ▶
Record Sales = Record Volume * Record Price
Record Volume = ◀ UNDEFINED ▶
Total Expenses = Overhead + Book Exepnses + Record Expenses
Total Sales = Book Sales + Record Sales
Overhead = 500
CONSTANT 500
87K
CALC
READY

Figure 3.2: Display of formula for a constant

Javelin displays the CONSTANT indicator to acknowledge that the value of Overhead does not change over time. Create Book Price and Record Price similarly.

Book Price = 7.50
Record Price = 12.50

Viewing a Table of Variable Values

■ You have now entered all the variables in your model design, with two exceptions: Book Volume and Record Volume. These are the only time-series input variables in the model. Each variable will have a series of values that you must enter. Although time-series data may be entered into Javelin in a number of ways, the most convenient is to display a table of values or a variable and to enter data in that table. To view a table of values for a variable, use the / **V**iew **T**able command: Press / to see the Main menu. Choose the View command either by pressing V or by moving the cursor to highlight View and then pressing ⟵. The View options menu appears (see Figure 3.3).

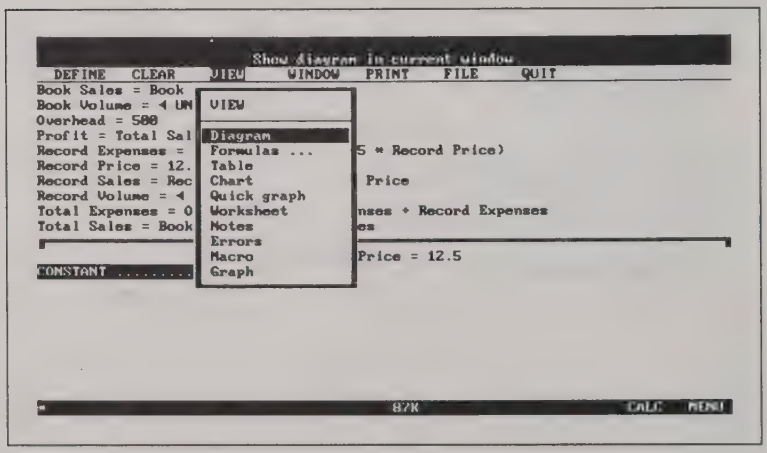


Figure 3.3: View options menu

You want to see a Table view of the Book Volume variable, which you achieve by selecting Table from the View menu. The message *Variable name (Record Price):* should appear at the top of the screen, and the comment *Show table of data for variable* should appear below it. You have told Javelin that you wish to view a table, and Javelin is asking what variable to show in that table. The last variable you defined was Record Price, so Javelin suggests it. But since you want to look at Book Volume you can override this suggestion by typing

Book Volume

Three columns of dates appear in the lower half of the screen. There are no values in the table yet (see Figure 3.4). The dates on your screen will probably be different. Since Javelin is guessing about what dates you are interested in, it starts your table from the current month and year.

Start your model at the beginning of 1987. Press ← and → until a column containing the date Jan 1987 appears and the cursor is in that column. If the first date on your screen is later than January 1987, move to the leftmost column and then move left again. A new column of dates will appear. Continue until a column containing Jan 1987 appears. Similarly, if the last date on your screen is earlier than January 1987, move the cursor to the right until a column with Jan 1987 in it appears. Using ↑ and ↓, position the cursor on Jan 1987, so that your screen resembles Figure 3.5.

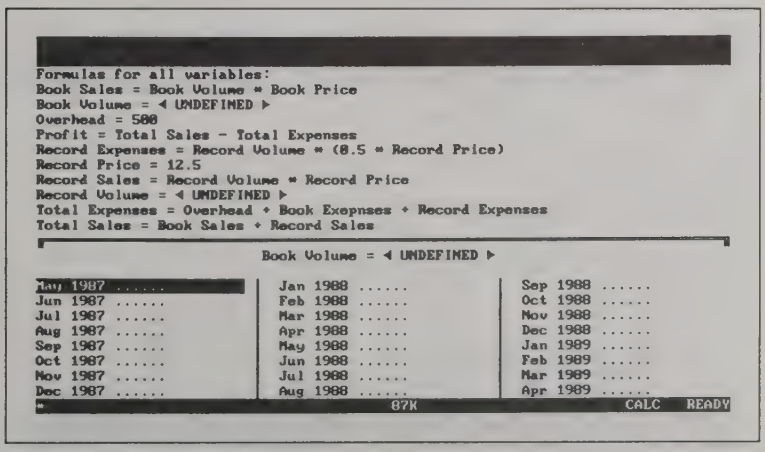

Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◄ UNDEFINED ►
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
Record Volume = ◄ UNDEFINED ►
Total Expenses = Overhead + Book Expenses + Record Expenses
Total Sales = Book Sales + Record Sales
Book Volume = ◄ UNDEFINED ►
May 1987 .....
Jun 1987 .....
Jul 1987 .....
Aug 1987 .....
Sep 1987 .....
Oct 1987 .....
Nov 1987 .....
Dec 1987 .....
Jan 1988 .....
Feb 1988 .....
Mar 1988 .....
Apr 1988 .....
May 1988 .....
Jun 1988 .....
Jul 1988 .....
Aug 1988 .....
Sep 1988 .....
Oct 1988 .....
Nov 1988 .....
Dec 1988 .....
Jan 1989 .....
Feb 1989 .....
Mar 1989 .....
Apr 1989 .....
87K
CALC
READY

Figure 3.4: Blank Table view

Entering Data

■ Now enter your first piece of data—the number of books you plan to sell in January. Type

```
25
```

The screen reflects the fact that you entered one value for Book Volume in a monthly table. Therefore, Javelin knows that this time-series variable contains monthly data and that, right now, the data start and end in January 1987. Also, pressing ⟵ has moved the cursor to Feb 1987.

Enter data from Figure 3.8 for the rest of 1987. If you make an error, use the cursor keys to position the cursor on the month you wish to change and then type the correct value. If you accidentally enter a number for a date beyond 1987, don't worry about it. You will find out how to remove data from (and insert data into) time series in Chapter 6, "Javelin Views and Windows." Your completed table should contain the values shown in Figure 3.6.

■ Creating Another Time-Series Variable

Finally, you must enter data for Record Volume. View the Record Volume table just as you did the Book Volume table. Make sure you have the

```
Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◄ UNDEFINED ►
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
Record Volume = ◄ UNDEFINED ►
Total Expenses = Overhead + Book Exepnses + Record Expenses
Total Sales = Book Sales + Record Sales

                    Book Volume = ◄ UNDEFINED ►

Sep 1986 ......      May 1987 ......      Jan 1988 ......
Oct 1986 ......      Jun 1987 ......      Feb 1988 ......
Nov 1986 ......      Jul 1987 ......      Mar 1988 ......
Dec 1986 ......      Aug 1987 ......      Apr 1988 ......
Jan 1987 ......      Sep 1987 ......      May 1988 ......
Feb 1987 ......      Oct 1987 ......      Jun 1988 ......
Mar 1987 ......      Nov 1987 ......      Jul 1988 ......
Apr 1987 ......      Dec 1987 ......      Aug 1988 ......
                          86K                    CALC   READY
```

Figure 3.5: Table view with cursor positioned on Jan 87

READY indicator at the bottom right of the screen, then issue the command **/ V**iew **T**able and type

Record Volume

Use the cursor keys as you did above to position the cursor on Jan 1987 and enter data so that your table contains the values shown in Figure 3.7.

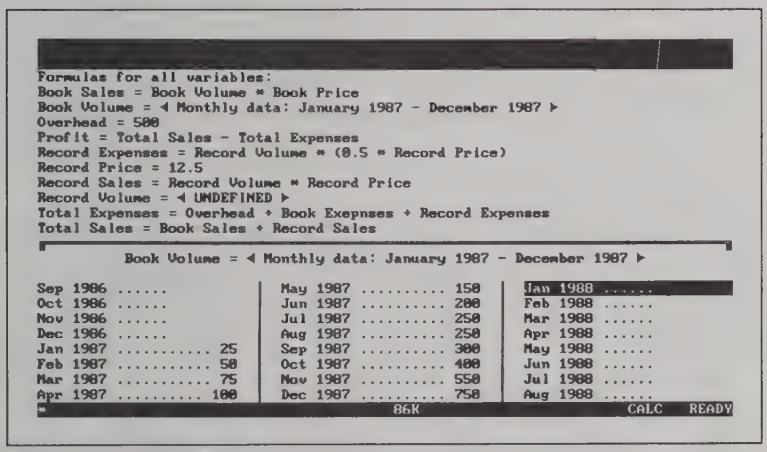

Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
Record Volume = ◄ UNDEFINED ►
Total Expenses = Overhead + Book Exepnses + Record Expenses
Total Sales = Book Sales + Record Sales
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Sep 1986 ..... | May 1987 ..... 150 | Jan 1988 .....
Oct 1986 ..... | Jun 1987 ..... 200 | Feb 1988 .....
Nov 1986 ..... | Jul 1987 ..... 250 | Mar 1988 .....
Dec 1986 ..... | Aug 1987 ..... 250 | Apr 1988 .....
Jan 1987 ..... 25 | Sep 1987 ..... 300 | May 1988 .....
Feb 1987 ..... 50 | Oct 1987 ..... 400 | Jun 1988 .....
Mar 1987 ..... 75 | Nov 1987 ..... 550 | Jul 1988 .....
Apr 1987 ..... 100 | Dec 1987 ..... 750 | Aug 1988 .....
86K
CALC
READY

Figure 3.6: Data for Book Volume table

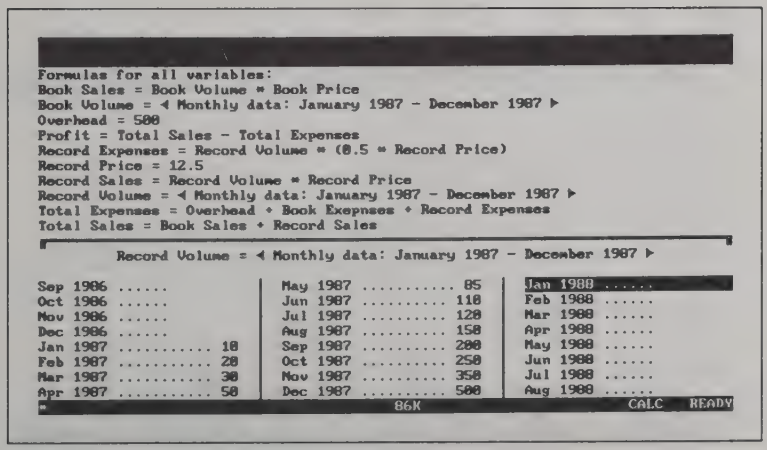

Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Total Expenses = Overhead + Book Exepnses + Record Expenses
Total Sales = Book Sales + Record Sales
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Sep 1986 ..... | May 1987 ..... 85 | Jan 1988 .....
Oct 1986 ..... | Jun 1987 ..... 110 | Feb 1988 .....
Nov 1986 ..... | Jul 1987 ..... 120 | Mar 1988 .....
Dec 1986 ..... | Aug 1987 ..... 150 | Apr 1988 .....
Jan 1987 ..... 10 | Sep 1987 ..... 200 | May 1988 .....
Feb 1987 ..... 20 | Oct 1987 ..... 250 | Jun 1988 .....
Mar 1987 ..... 30 | Nov 1987 ..... 350 | Jul 1988 .....
Apr 1987 ..... 50 | Dec 1987 ..... 500 | Aug 1988 .....
86K
CALC
READY

Figure 3.7: Data for Record Volume table

Congratulations—you have now completed three-fourths of your first model. You have taken an idea, designed a model to describe it, and entered this model into Javelin. All that's left to do is find the answer, given your current assumptions.

Looking at the Answer

■ First, look at the Table view of the Profit figures, which you create by the same procedure used to enter the Book Volume and Record Volume data. Give the command / **V**iew **T**able and type

Profit

What appears is identical to the Profit formula in Figure 3.2, indicating that profit values have not yet been calculated.

Originally, this made sense—you had just typed in the Profit formula and neither Total Sales nor Total Expenses had been defined. But now you have typed in all the necessary information, and Profit is still undefined. Why? The CALC indicator appeared on the status line when you typed in your first formula and has been there ever since, indicating that changes have been made to the model since it was last recalculated. To tell Javelin to recalculate your model, press F9, the Calc key. Your Profit table should now look like Figure 3.8.

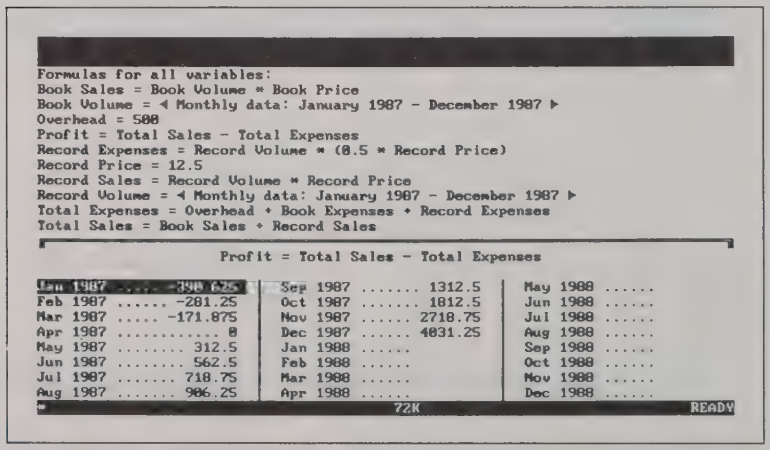

Formulas for all variables:
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Total Expenses = Overhead + Book Expenses + Record Expenses
Total Sales = Book Sales + Record Sales
Profit = Total Sales - Total Expenses
Jan 1987 ..... -390.625
Feb 1987 ..... -281.25
Mar 1987 ..... -171.875
Apr 1987 0
May 1987 ..... 312.5
Jun 1987 ..... 562.5
Jul 1987 ..... 718.75
Aug 1987 ..... 906.25
Sep 1987 ..... 1312.5
Oct 1987 ..... 1812.5
Nov 1987 ..... 2718.75
Dec 1987 ..... 4031.25
Jan 1988 .....
Feb 1988 .....
Mar 1988 .....
Apr 1988 .....
May 1988 .....
Jun 1988 .....
Jul 1988 .....
Aug 1988 .....
Sep 1988 .....
Oct 1988 .....
Nov 1988 .....
Dec 1988 .....
72K
READY

Figure 3.8: Profit table after calculation

Javelin now knows what time periods to display for Profit; since only the months in 1987 have sales data, those are the only months for which Javelin can calculate Profit figures and the only months it displays.

So far you have looked at two Javelin views: the Formulas view, which shows a list of your model's formulas, and the Table view. Javelin's power is in its ten different views, each of which is useful for a different facet of working with a model. While you can enter formulas and data in almost any view, the Worksheet view is by far the most powerful view with which to look at your model.

Creating a Report

■ Leaving the Profit table on the screen, compare book and record sales and expenses for all of 1987. To do so, leave the bottom window showing the Profit table and work in the top window.

To make the top window active, press F6, the Window key. The cursor, which was highlighting Jan 1987 in the bottom window, now moves to the top window.

■ *Creating a Worksheet*

You are going to create a worksheet called Comparison. To do so, give the command / **V**iew **W**orksheet and name it by typing

Comparison

A blank worksheet, similar to a spreadsheet, appears in the upper window (see Figure 3.9). Your cursor is a highlighted block (on graphics displays) or an underline (on nongraphics displays).

You need only enter a few items to tell Javelin what you wish to see. The cursor is in the upper-left corner of the worksheet. (You may move it there at any time by pressing Home.) To compare sales and expenses information for your record and book products in 1987, enter

1987

in the upper-left corner of the worksheet border.

In most spreadsheet programs, this area is blank, while the border row to its right is reserved for column letters (such as A, B, C ... AA, AB) and the

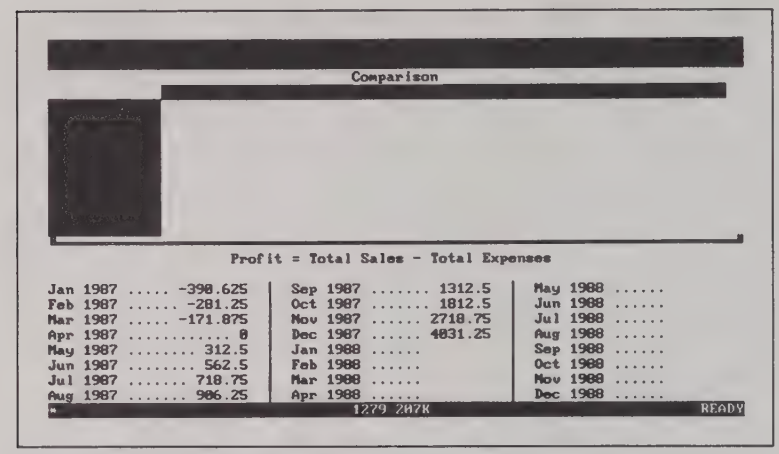


Figure 3.9: Blank worksheet

border column below it is reserved for row numbers (1 through 8192). Not so in Javelin: The upper-left corner of a worksheet is a *filter* indicating what data from a model (in this case, which time period) this worksheet should refer to. The top and side borders tell Javelin what variables you want on the worksheet.

For example, move down to a blank row in the worksheet by pressing ↓ twice. Now enter

Sales

since this is the information you want to see on this row. Move down another couple of lines and tell Javelin you want expenses information on this line. Type

Expenses

Your worksheet should now look like Figure 3.10.

Now, to tell Javelin what sales and expenses figures you are interested in, go back to the upper-left corner of the worksheet (by pressing Home) and move to the first column of the worksheet (by pressing →).

In this column you will show book information, so type

Book

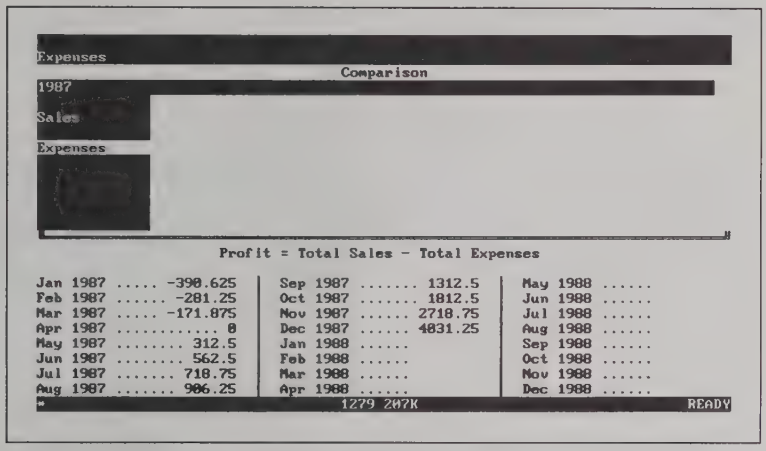


Figure 3.10: Comparison worksheet with left border

Book Expenses and Book Sales for 1987 now appear on the worksheet. Javelin combines the title you just typed (Book) with each of the titles along the left border (Sales and Expenses) and comes up with Book Sales and Book Expenses. It then reads the upper-left corner and fills in the information for Book Sales and Book Expenses corresponding to that time period, 1987.

To generate the same information for the record product line, move right a couple of columns and type

Record

Javelin fills in the information for Record Expenses and Record Sales for 1987 as well (see Figure 3.11).

■ *Creating Another Worksheet*

Perhaps this has whetted your appetite for a more complete profit-and-loss outlook of the projected business. Javelin allows you to set up as many worksheets as you like, each containing a different layout of information from the model. To build a profit-and-loss worksheet in the bottom window, first move to that window with the Window key, F6.

The cursor stops highlighting Record in the top window and should highlight Jan 1987 in the bottom window. Now, give the command / **V**iew

Worksheet and give the worksheet a name, like

P&L

Another blank worksheet appears. In this worksheet, you can view your information by quarters. Unlike the Comparison worksheet above, this one will refer to several time periods. Therefore, leave the upper-left corner blank. Move right to the first column and, because you want to see information for the first quarter, type

1Q87

Javelin understands that this refers to the first quarter of 1987 and reformats what you typed into *1 Q 1987*. To enter the column dates for the next quarter, simply press → and ⟵. Instantly, *2 Q 1987* appears. Just pressing ⟵ is a shorthand way to tell Javelin to continue a series of dates: since your last date was a quarter, Javelin fills in the date for the next quarter and moves to the right, ready to fill in the next quarter. So, to enter headings for quarters three and four, merely press ⟵ twice.

Now tell Javelin to give you overall figures for the year in the last column by typing

1987

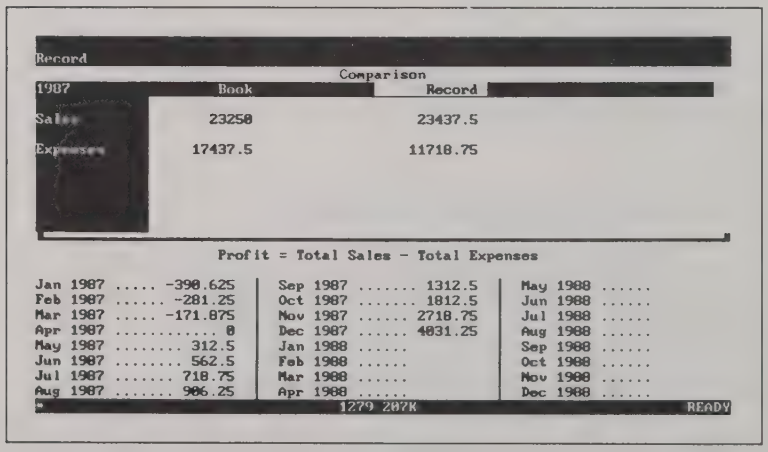


Figure 3.11: Completed Comparison worksheet

Your screen should look like Figure 3.12; if it doesn't, move to the appropriate column heading and type the proper information.

So far, no data appear in your worksheet because Javelin does not know what variables you want for these quarters. You need to enter variable names in the left border of the worksheet. Look at Book Sales and Record Sales first, then expenses for them. To make your worksheet clearer, make one heading for Sales, and list Book and Record sales beneath it. To do this in Javelin, first you need to go back to the left border of the worksheet. The fastest way to get there is to go Home to the upper-left corner of the worksheet and then move down. This positions you in the left border of the worksheet's first row, where you can tell Javelin which variable to place on that row. Type

Sales

The word Sales appears, but nothing else happens. This is because there are two variables in your model whose names contain that word: Book Sales and Record Sales. Since you did not type a full variable name, Javelin assumes that the word Sales is a heading, and, *assuming you indent the subheadings,* Javelin will combine the headings and subheadings to tell it what variables to use. This is similar to what it did in the top worksheet, combining words from the top and side borders to determine what information to show. This time, it combines the heading (Sales) with the subheadings you

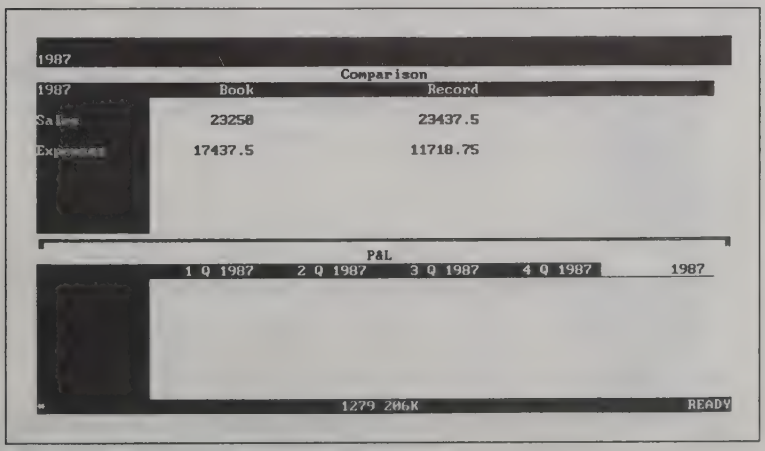


Figure 3.12: Headings for P&L worksheet

are about to add. Move down to the next row and type one space, then

Book

The figures for Book Sales should appear on your worksheet. If not, you probably forgot to type the space in front of Book. That space is important because it tells Javelin to combine the words Book and Sales to form a variable name. Next, enter the Record subheading. Type a space, then

Record

Finally, add a line for Total Sales, separated from the detail by a blank line. Press ↓ twice and type

Total Sales

Your screen should look like Figure 3.13.

Now do the same for Expenses. Type

Expenses

Remember that you have one additional expense, Overhead. First press ↓

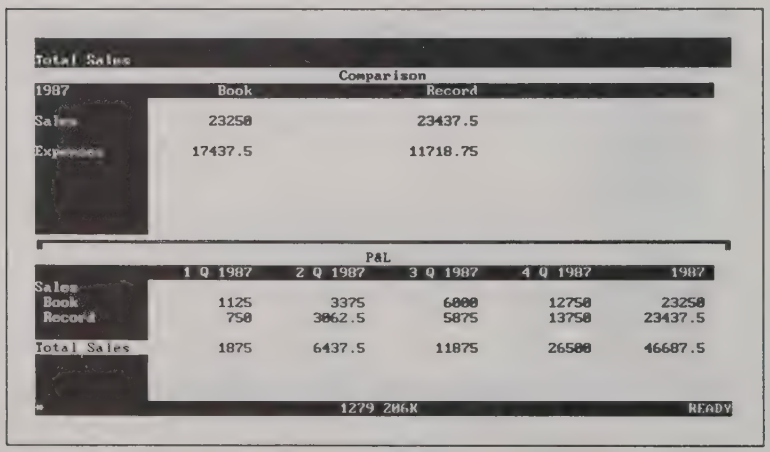


Figure 3.13: P&L worksheet showing sales data

twice, then type the following (press ↓ and add the leading space for each entry):

Book
Record
Overhead

Press ↓ twice and type

Total Expenses

When you pressed the ↓ key before you entered Record (for Record Expenses), you may have noticed that the word Sales disappeared off the top of the window. Like most spreadsheet and word-processing programs, Javelin automatically *scrolls* information off the screen (or window) if it will not fit. And the title Total Expenses actually appears as Total Expens because the full title will not fit in the left border (later chapters will discuss how to change the sizes of windows and borders).

Finally, put in a row showing profit, by moving down twice and typing

Profit

The Expenses part should look like Figure 3.14.

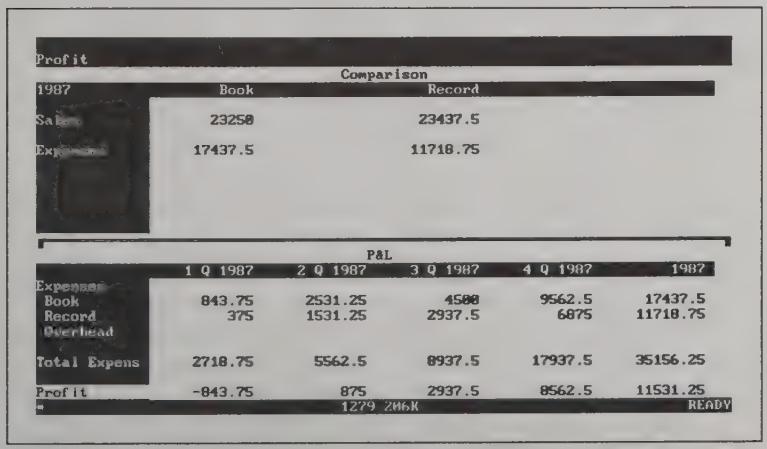


Figure 3.14: P&L worksheet showing expenses and profit

Your sample Profit and Loss statement is complete, although you should do two things to be really finished. The first is to save the model on disk so that you can continue to build on it later; the second is to print the statement.

Saving the Model

■ The asterisk indicating that changes have been made to the model since it was saved has appeared at the left side of the status line. (This happened earlier, when you typed the first item of the model.) To save the model, press / and select **F**ile.

The File menu appears. In this case, you want to save the model on the disk, so select **S**ave. On the top line of the screen (the input line) Javelin asks for a name under which to save your model. (Names can be up to eight characters long, and can be a mixture of letters and numbers, without spaces.) Let's call this model BOOKRECD (for Books and Records); just type the name and press ⟵. Javelin automatically adds the letters *.MDL* to the name you type. This extension identifies the disk file BOOKRECD.MDL as a Javelin model. The file name should appear in the lower-left corner of your screen, replacing the asterisk.

Printing a Worksheet

■ The goal of creating a model is usually to produce reports. Printing in Javelin is done through a command called (not surprisingly) Print. Make sure that your printer is on, is "on line," and has paper in it. To bring up the Print menu, press / and select **P**rint. To print your P&L statement as it currently exists, select **C**urrent view from the menu.

An options menu appears, like those described in Chapter 2. The question, Print Borders on Each Page?, always appears, although it applies only to worksheets more than one page long. Although this worksheet is very short, the / **P**rint **C**urrent view command automatically divides a longer worksheet and prints it on as many pages as necessary. Since your worksheet will fit on one page, your answer doesn't matter. So select **Y**es and printing should commence.

As with spreadsheet programs, it is best to let the program advance the paper in the printer so that it keeps track of where the page actually ends. This becomes important when your view will not fit on one page; Javelin then divides it among a number of pages, but only if it knows where the paper really ends. To roll your model out of the printer, tell Javelin to eject a page or two by using the command / **P**rint **P**age. You may now tear off and appreciate the output of your model.

Javelin allows you to print any view, thereby producing lists of formulas, tables of values for each variable, and other ways to document your model. For fancier formatting of your reports, see Chapter 13, "Advanced Worksheets and Printing."

Conclusion

■ This model is much simpler than the models you probably have in mind, but it contains all the elements you will need to create most models: formulas, data, and constants. You've seen how to use information from your model to create a simple report in the Worksheet view. You can elaborate on this model to make it more true to life (for instance, by dividing up overhead into its different components, dividing sales geographically, or putting in different prices for different books).

But before you do so, read on. In the next two chapters, we explain additional features that give you more control over Javelin—using settings sheets as well as more commands. In future chapters, we will discuss each feature introduced here in much more detail.

CHAPTER FOUR

Managing Files and Printing Views

Two sets of commands you will need every time you run Javelin are those for managing files and printing views—you saw them briefly in Chapter 3. This chapter discusses in detail the / **F**ile and / **P**rint commands and their options. It begins by describing basic file management operations in DOS, the program that runs behind the scenes and handles all communication with disks. If you're already familiar with DOS you may want to skip ahead to "Javelin's Default Directory." For full information about DOS, read the DOS manual that came with your PC. Or you may want to read the *MS-DOS Handbook* by Richard Allen King (Berkeley: SYBEX, 1985) or *Running MS-DOS* by Van Wolverton (Seattle: Microsoft Press, 1985). Here is an overview of what you need to know about DOS to use Javelin.

File Operations

■ When you are working on a model, all the variables, formulas, settings, graphs, and worksheets are stored in the random-access memory (RAM) of the PC. The RAM serves as the work area for the program and data you are running. Since the contents of this memory are lost when you exit Javelin or turn off the PC, you must copy the model onto a disk to be stored more permanently.

■ File Names

DOS stores each set of information on a disk as a file, with a *file name*. A DOS file name has three parts: the *drive name*, the *name*, and the *extension*. If you use a hard disk with subdirectories, there may be a fourth part, the *path name*.

The drive name tells where the file is stored—which disk drive the information is on. The PC's diskette drives are called *A:* and *B:* (if you only have one, it can be referred to as either *A:* or *B:*). If your PC has a hard disk, it is called *C:*. If you have other drives, they may be called *D:* and so on. A colon always follows the drive letter.

The name itself can be up to eight letters long, with no spaces or punctuation. Make the name as descriptive as possible: BUDGET87 or SALES3 is easier to remember than XU8E45.

The extension can be up to three letters long and is separated from the file name by a period. The extension, which is optional, indicates what kind of file this is. For Javelin models, the extension is always *MDL,* for *model.*

The three parts of the file name are strung together with no spaces between them. Here are some examples of file names:

- A:BUDGET.MDL (model called BUDGET on drive A:)
- C:CARSALES.MDL (model called CARSALES on drive C:)

■ *Path Names and Subdirectories*

A diskette holds a relatively limited amount of information, from one to about twenty models (depending on the size of the models). A hard disk can hold thirty to one hundred times as much data as a diskette. To keep track of what is stored on your hard disk, you should set up *subdirectories,* which are like separate areas on the disk. You can have one subdirectory for DOS programs, one for the Javelin program itself, and one for your Javelin models. If you have many Javelin models, you can set up several different directories for your models—one for budgets, one for personnel tracking, and for other related models.

Each disk, whether it is a diskette or a hard disk, has a main directory, or main area where files are stored, called the *root* directory. This name comes by an analogy with a tree, whose base is the root. In DOS, the name of the root directory is abbreviated by a single character: \.

In the root directory you can store files and also other directories. For example, you can create other directories in your root directory so that your hard disk looks like Figure 4.1. This is called a *tree-structured directory* (although the tree is upside down, with the root at the top, you can see the structural similarity). In this example, there are four directories: the root, DOS, JAV, and MODELS.

In the DOS directory, you can store the programs that come with DOS, which include about two diskettes full of programs, such as FORMAT, DISKCOPY, and other commands you may use. The JAV directory contains the Javelin program itself—Javelin comes on five diskettes, containing the program itself, configuration information, sample models, and other files. The MODELS directory should contain your models. This way, when you look at the list of files in your MODELS directory, it is not cluttered up with programs, word-processing documents, and other extraneous files.

Your subdirectories can contain subsubdirectories, creating another level of the tree, but most people find that this gets too confusing. The idea of using subdirectories is to make it easier to find and use all your files on a hard disk, so don't make the structure of the directories too complicated.

If you have created subdirectories, you can include the name of the directory in the file name, between the drive name and the name of the file. A file name that indicates what directory a file is in is called a *path name*—the instructions, or path, for getting from the root directory of the disk drive down to your particular file. A path name usually starts at the root, so the first character in the path name is a backslash: \. Then you indicate which directory to go to. As you list the steps for DOS to follow to find your file, add another \ every time you drop down to the next subdirectory.

For example, if you have a model called BOOKS.MDL stored in the MODELS subdirectory on drive C:, the path name would be:

C:\MODELS\BOOKS.MDL

Here are the parts of this path name:

C:—Disk drive
\—Start at the root directory
MODELS—MODELS subdirectory off the root
\—Drop down into the MODELS subdirectory
BOOKS.MDL—Name of the file

In your DOS manual, you can find information about commands to create a directory (MKDIR), use directories (CHDIR, PATH, and TREE), and remove a directory (RMDIR).

When you install Javelin on a hard disk, the installation program automatically creates a subdirectory in the root called JAV and stores the Javelin program files in this directory.

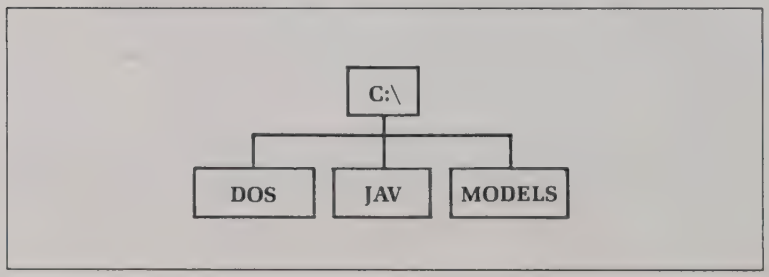


Figure 4.1: Tree-structured directory

■ Javelin's Default Directory

When you install Javelin, it has a *default disk drive* and *default directory* in which it assumes that you want all models to be stored. When you give a command that refers to a file on a disk, it looks in that directory on that disk drive. You can change these defaults by using the command / **D**efine **C**onfiguration to change the Configuration settings sheet, described in Chapter 5, "Special Keys and Settings Sheets."

Using Javelin's / File Commands

■ The / **F**ile commands let you store a model from memory onto a diskette or a hard disk, load it back into memory, erase a model you no longer want, or import information from other files into the model you are working on. Table 4.1 lists the choices on the File menu.

■ Loading a File from the Disk

Use the / **F**ile **L**oad command to read in a model file. To load the BOOKRECD model you created in Chapter 3, give the command / **F**ile. You see the File options menu, which looks like Figure 4.2.

Choice	Description
Load	Loads a model from the disk into memory, clearing the model currently in memory.
Save	Saves the model in memory on the disk.
Import	Loads part of a model on the disk into the model currently in memory, without clearing the model in memory. You can import a whole model or just part of it (see Chapter 18, "Transferring Information Between Models").
Delete	Deletes a file on the disk.
Run DOS	(Javelin PLUS) Runs DOS or another program. To return from using DOS, give the DOS command EXIT. Because Javelin and your model remain in memory, there is not enough free RAM to run most programs. Do not run or release memory-resident programs, like DOS MODE or PRINT, or SideKick.

Table 4.1: Choices on the File menu

Now select **L**oad. Tell Javelin which model you want by typing

BOOKRECD

You do not have to type in the MDL extension; Javelin adds that automatically when you press ⏎. It doesn't matter if you use small or capital letters to type in the file name. The model should appear on the screen just as it looked when you saved it in Chapter 3. The same views are on the screen, the cursor is in the same place, and everything is just as you left it.

If you were now to load a different model, Javelin would erase this model and replace it with the new one. You should always save one model before loading the next.

■ *Possible Problems Loading a File*

The most common reason Javelin cannot load a file you ask for is that you have asked for a nonexistent file or a file that is not in the current directory. If you ask for a file Javelin cannot find, it responds with the message *Unable to load model.mdl; File not found.* Check the name of the file and the directory where Javelin is looking for that file.

A second reason Javelin may not be able to load a model is that the model may not fit in memory. The most common instances of this are when you receive a model from someone whose computer has more memory

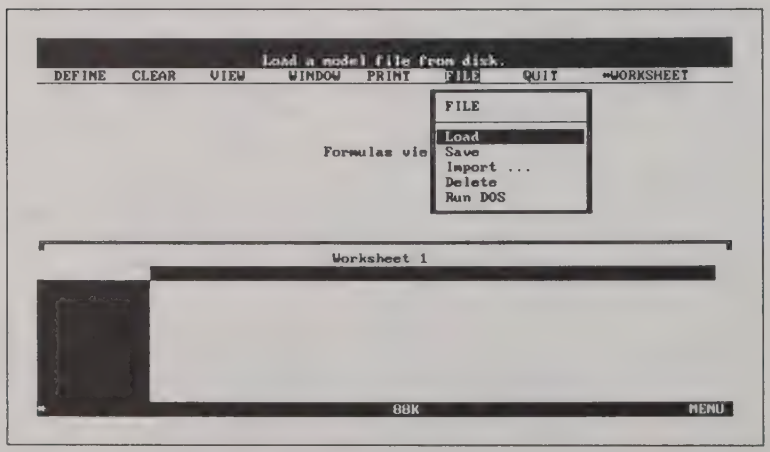


Figure 4.2: File options menu

than yours, or when you add memory-resident software to your computer. Whatever the cause, if Javelin runs out of memory while loading your model, you get the message *model.mdl partially loaded; Not enough memory.* You may also see part of your model on the screen. *Do not use this model* unless absolutely necessary! It may be missing formulas or data, and if you save it, you will destroy the existing complete copy of your model, which is still on disk. Find out why there is not enough memory (such as the presence of new software or removal of a memory board for repairs) or find a machine with more memory before you continue.

■ *Saving a Model on a Disk*

When you are done working on a model, you must save it onto a disk before working on another model, quitting Javelin, or turning off the PC. The command is / **F**ile **S**ave. In response, Javelin asks you for a name. You type in the name (up to eight letters) and press ⟵. Javelin automatically adds the file extension MDL to the name and saves it on the disk, in the default directory on the default disk drive.

Save the model you are working on with the name TEST (make sure that you are in Ready mode before you start the command). Press /, select **F**ile, select **S**ave, and press ⟵. Javelin prompts you for the name of the model. Type

TEST

and press ⟵. The red light flashes on the default disk drive, showing that the file is being written onto the disk. Javelin confirms that the file has been saved with the message *Test.mdl saved* on the information line. The model's name on the status line changes to TEST.MDL. The model is still in memory—/ **F**ile **S**ave copies the model into a file on the disk without erasing what is in memory.

■ *Saving a File a Second Time*

If you save a model and continue to work on it, an asterisk after the model's name on the status line indicates that you have made changes to the model in memory and that the changes should be saved before you quit Javelin. The next time you give the / **F**ile **S**ave command, when Javelin prompts you for the file name it suggests the same name the model had the last time you saved it. To use this name again, just press ⟵.

The screen goes back to the way it looks when you first load Javelin. If you hadn't saved the model before clearing, you would have lost all your work.

Javelin then tells you that there is already a file on the disk with that name and asks if you want to erase the previous version of the file and replace it with this new version. You can choose to go ahead and replace it, or you can cancel the Save command and try again with another file name.

Let's try it: give the / **F**ile **S**ave command. Javelin asks for the file name, suggesting TEST. Save it using the same name as before.

A *query menu* will appear, asking you to confirm that you want to overwrite the previous TEST.MDL on the disk, as shown in Figure 4.3. Javelin has many query menus that "pop up" to ask you a question. Choices are listed beneath the question, and you move the cursor to the one you want and press ⟵ to select it, just like on a normal Javelin menu. Select **Y**es to save it.

■ *Clearing a Model in Memory*

After you have saved a model, you may want to erase it from memory so that Javelin contains no model. This allows you to begin building a new model in memory.

The command to clear an entire model from memory is / **C**lear **A**ll. Use it now, and soon you'll reload the model you saved. You can select other choices on the Clear menu to clear only certain parts of the model; Table 4.2 lists the choices.

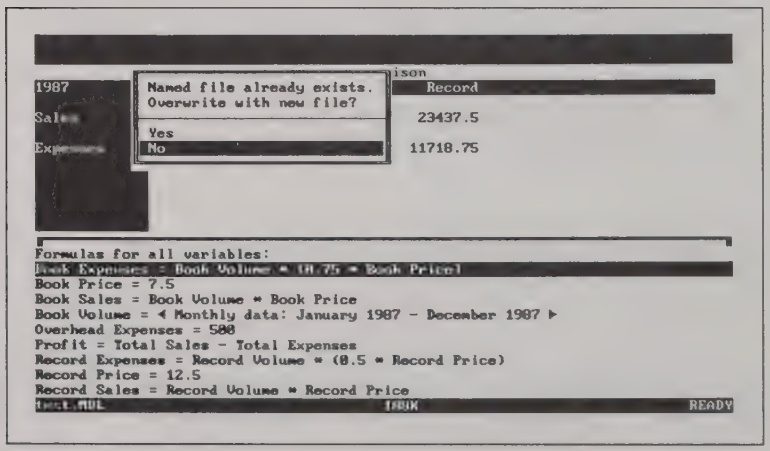

ison
1987
Named file already exists.
Overwrite with new file?
Record
Sales
23437.5
Yes
No
11718.75
Formulas for all variables:
Book Expenses = Book Volume * (0.75 * Book Price)
Book Price = 7.5
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead Expenses = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Record Sales = Record Volume * Record Price
INBOX
READY

Figure 4.3: Query menu for overwriting a file

Save early and often. At any moment, you, a co-worker, or your dog could knock the plug out of the wall and you would lose all your work if you hadn't saved it. It's a good idea to save about every 20 minutes, so that if disaster strikes or you give the / Clear All command by mistake, you won't lose more than 20 minutes' worth of work.

■ Possible Problems Saving a File

If there is not enough room on the diskette or hard disk for the model to be saved, Javelin displays the message *Not enough disk space to save file.* At this point, you can delete files you no longer need (see below) and try saving again. If you are using diskettes, you can try a different diskette.

If you type in a path name, like C:\MODEL\TEST, and you have misspelled the name of the directory (perhaps it should be MODELS), you may get the error message: *Directory not found.* If you use a strange character in

Choice	Description
Variable	Deletes a variable from the model. If the variable is referred to in other formulas, the variable remains in the model but is undefined.
Building block	Deletes a building block (see Chapter 17, "Building Blocks").
Worksheet	Deletes a worksheet and its labels, border entries, exception cells, and range calculations. No variables are deleted (see Chapter 7, "Defining Variables and Entering Data Using Worksheets").
Graph	Deletes a graph and its settings, but no variables are deleted (see Chapter 12, "Graphing").
Macro	Deletes a macro (see Chapter 14, "Macros").
Formula	(Javelin 1.1 and later.) Converts a variable from a calculated variable to an input variable, using the current values of its formula.
Data	Deletes the data for all variables for periods outside the two dates you specify.
Note	Deletes a note (see Chapter 6, "Javelin Views and Windows").
All	Deletes the entire model from memory. Save your model on disk first if you do not want to lose your work.

Table 4.2: Choices on the Clear menu

the file name, such as a space or a punctuation mark, you get the message *Invalid file name.* Use only letters and numbers in the name, with no spaces.

■ Deleting Files

If you no longer want to keep a model on the disk, you can delete the file containing it by using the / **F**ile **D**elete command. You can also delete other kinds of files.

Try giving that command to delete the TEST model you saved earlier. You are asked which file you want to delete. Type in the full name of the file, including the extension (MDL):

TEST.MDL

Javelin gives you a last chance to reconsider—a small query menu pops onto the screen, asking you to confirm that you want to delete the file. Select **Y**es.

Javelin confirms that the file was deleted. Always be very careful when deleting files—without specialized utility programs, there is no way to get back a file once you have deleted it.

■ Running DOS from Within Javelin PLUS

In Javelin PLUS, you can exit to DOS without losing your model. The / **F**ile **R**un DOS command gives you a *small* DOS work area, suitable for formatting a new diskette, creating a new subdirectory on your hard disk, or backing up files from your hard disk. The most important use for this feature is to allow you to make space if for some reason you can't save your model.

There are two ways to use this command: when you give the / **F**ile **R**un DOS command, Javelin gives you the opportunity to enter a DOS command. If you just want to do one thing (like format a diskette), you can type the command here. Javelin will ask DOS to execute the command and come right back to Javelin when the command completes. If you want to type several commands to DOS, hit ↵. You will see the familiar DOS prompt. Use DOS normally. When you are finished using DOS, use the EXIT command to return to Javelin.

This DOS window does *not* make Javelin a memory-resident program; you should not, as a rule, load Javelin, give the / **F**ile **R**un DOS command, and run other major applications. For one thing, the DOS you get has only about 100K of memory. The rest is used by Javelin and by your model. For

another, if the programs you run are not "well behaved" they could write on top of your model in memory, causing DOS and Javelin to crash and taking the model in memory with them.

Another caveat: do not load any memory-resident program while you are in the middle of using / **F**ile **R**un DOS. This includes the DOS PRINT, ASSIGN, GRAPHICS, GRAFTABL, and MODE commands as well as programs like SideKick and Metro. All these commands leave a little portion of themselves behind in memory after they run. Since they are running under Javelin, the results may be catastrophic. If you need to use a memory-resident program, load it *before* you run Javelin, so that the memory-resident part of the program is already in memory when Javelin is loaded.

You can try this now. Give the command / **F**ile **R**un DOS, and press ↵ when it asks for the DOS command. You'll see the same DOS prompt you saw before you started Javelin. Type a command like

DIR

When you are ready to return to Javelin, type

EXIT

Printing Information from a Model

■ Once you have entered information into a model, you probably will want to print it. If the BOOKRECD model is not loaded, load it with the / **F**ile **L**oad command. Select the Javelin command / **P**rint to bring up the Print menu, as you did when you printed the P&L statement in Chapter 3. The Print menu is shown in Figure 4.4, and Table 4.3 lists the choices on the menu and what they do.

■ *Aligning the Top of the Page*

Before you start to print, it is a good idea to make sure that Javelin knows where you are on the page—that is, if the printer is starting at the top of the physical piece of paper. To do this, use the / **P**rint **A**lign page command.

First, use the platen knob or other printer controls to move the paper so that the next thing you print is right at the top of the paper. Then select Align page to tell Javelin that your printer is at the top of the page. Your printer will probably reset—that is, the print head will jump back to the left margin and the printer's internal settings will be the same as if the printer were just turned on.

Javelin leaves top and bottom margins and keeps track of where it is on the page by counting lines. Printers normally print 6 lines to an inch, so a normal 11-inch page can hold 66 lines. As Javelin prints, it counts the lines, and when it gets to the bottom of a page, it leaves blank lines for the bottom margin of that page and for the top margin of the next page. If you move the paper yourself by using the Line Feed button on the printer or by moving the platen manually, Javelin won't know this and therefore its line count will be off. The symptoms of this are blank bands in the middle of a page—Javelin thinks it is skipping over the end of a page. If this happens, move the paper to the top of the next page and use the / **P**rint **A**lign page command to get the paper and Javelin back into sync.

■ *Printing the Current View*

To print the view that the cursor is in, use the command / **P**rint **C**urrent view. If you are working in a Formulas view, the printout shows the list of formulas. If you are in a Table view, Javelin prints the table of values for the variable you are looking at on the screen. This command prints the whole view, not just what you can see on the screen.

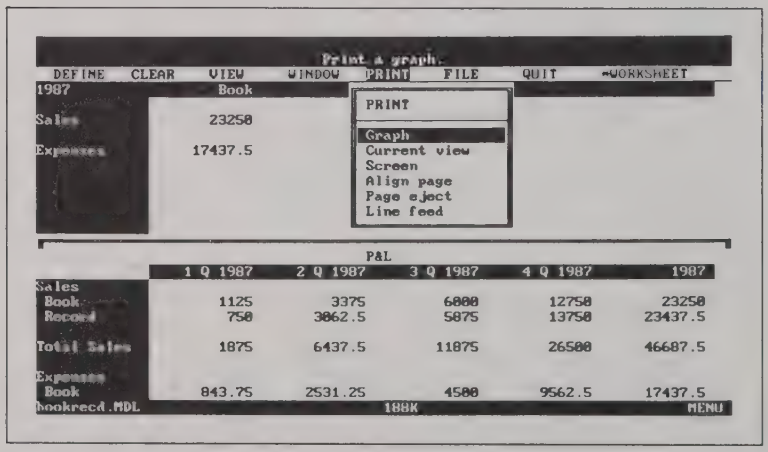


Figure 4.4: The Print menu

At the end of Chapter 3, you printed the P&L worksheet that is in the bottom window. As an example of how you can print other views, print out the list of all formulas in the BOOKRECD model. To do this, you must first bring that list onto the screen by selecting the command / **V**iew **F**ormulas **A**ll.

The P&L worksheet disappears from the bottom window and is replaced by a list of all the formulas in your model. (The / **V**iew **F**ormulas command is discussed fully in Chapter 6, "Javelin Views and Windows.") To print this list, give the command / **P**rint **C**urrent view. The whole list is printed, not just what fits on the screen.

■ *Printing the Contents of the Screen*

There are two ways to print only what's on the screen. To see one way, give the command / **P**rint **S**creen. This prints the view or views that are currently on the screen, without the edit line, information line, or status line. This command also allows you to print the contents of the screen into a file; see Chapter 13, "Advanced Worksheets and Printing," for more information on printing to a file.

Another way is to use a key built into the PC and DOS—the PrtSc key. Normally, this key produces an asterisk (notice the asterisk on the bottom half of the key). But when you hold down the Shift key and press PrtSc, you get a printout of the contents of the screen including the edit line, the information line, one or two views, and the status line. Try this now.

Choice	**Description**
Graph	Prints a graph you have already defined using / **D**efine **G**raph (see Chapter 12, "Graphing").
Current view	Prints the view you are currently working on, including parts of the view that may be off the screen.
Screen	Prints only the visible parts of the view or views that are currently on the screen.
Align page	Tells Javelin that the printer is at the top of a page.
Page eject	Ejects the current page and positions the printer at the top of the next one.
Line feed	Skips a line.

Table 4.3: Choices on the Print menu

You can use the PrtSc key when you are in the middle of giving a command, which makes it generally more useful than the / **P**rint **S**creen command.

■ *Moving the Paper*

To eject the page from the printer, give the command / **P**rint **P**age eject. This moves the paper in the printer to the top of the next page. Give this command if necessary to get the page out of the printer. You can also move the paper in the printer by using the / **P**rint **L**ine feed command, which sends a code to the printer to advance the paper one line.

Remember, do *not* advance the paper by using the knob on the form-feed and line-feed controls on your printer, because Javelin will get confused about where it should put the page breaks.

■ *Quitting Javelin*

When you are done using Javelin, you quit by giving the /**Q**uit command. Javelin asks you to confirm by selecting Yes or No. If you selected Quit by mistake, you can select No to remain in Javelin. If you select Yes, but you have changed your model since the last time you saved it, Javelin protects you from losing your work by pointing this out to you.

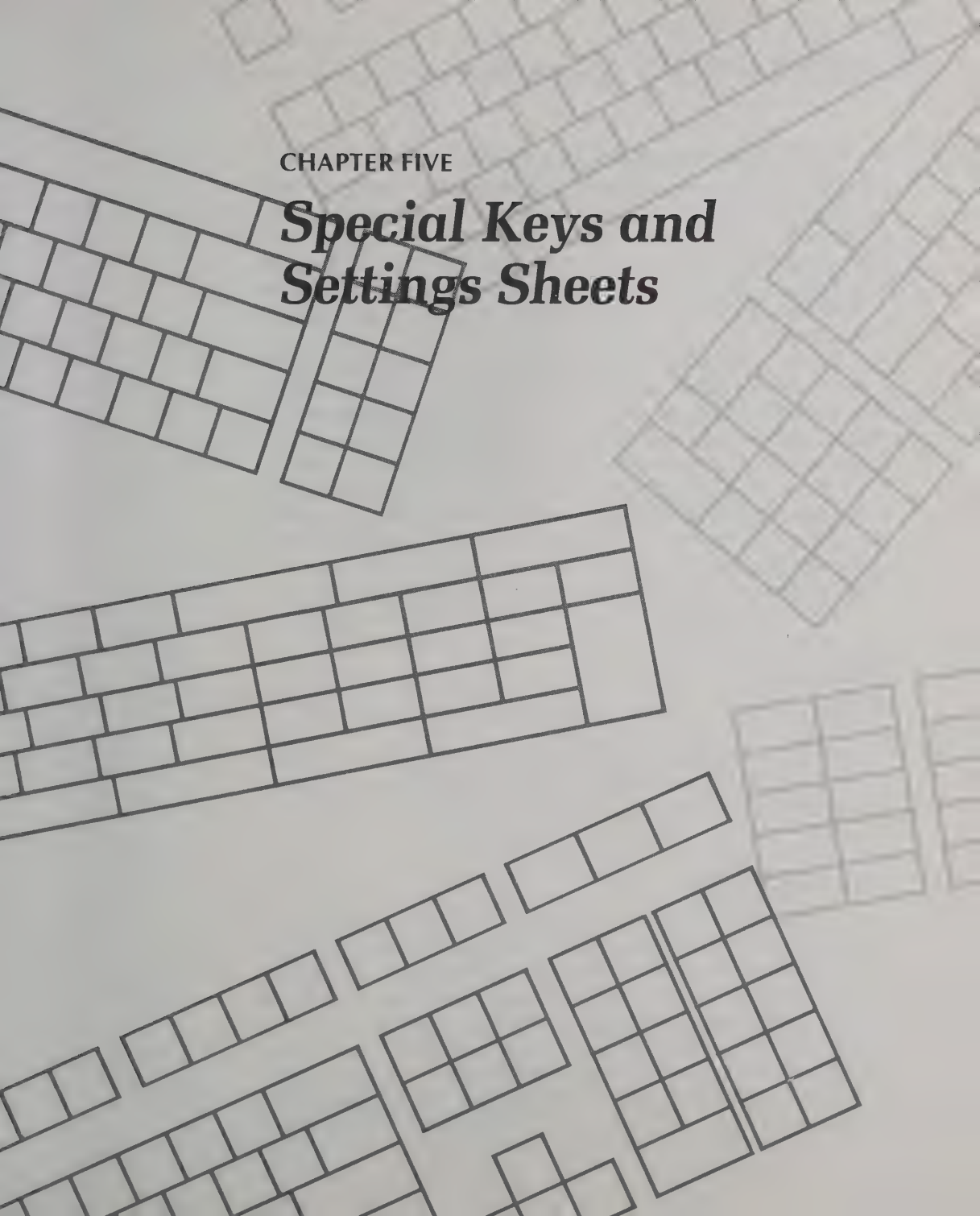
You may want to make a habit of using the **S**ave & Quit option on the Quit menu. Javelin will ask you what name you want to save your model under, just as it does for the / **F**ile **S**ave command. Use this option to exit from Javelin now. Give the / **F**ile **S**ave & Quit command, and hit ⟵. Select **Y**es when Javelin asks you if you want to replace your existing copy of the model with this new one.

Conclusion

■ In addition to giving commands such as / **P**rint and / **F**ile, another way to give Javelin instructions is by using settings sheets. We'll discuss these, as well as several time-saving keys, in the next chapter.

CHAPTER FIVE

Special Keys and Settings Sheets



There are three keys you will want to use in all Javelin views to edit information you have entered, bring variables into view, and find the possible answers to a Javelin prompt. The Edit key enables you to change formulas and text without retyping them. With the Goto key, you can move around in a view more directly than with the ↑ and ↓ keys. And the Spell key, which saves typing, may become your favorite key on the keyboard. There are additional keys you will use to move the cue, to insert and delete characters and words, and to cancel the editing process.

Javelin uses a unique method—settings sheets—to allow you to control the settings that define variables, worksheets, graphs, and other elements that make up a model. You will also use settings sheets for defining how numbers should be displayed, periodicity and time span, and other aspects of your model.

You will use the book and record model you developed in Chapter 3 to practice using these keys and settings sheets. Once you understand the commands and concepts in this chapter, you'll be ready to move into more advanced model design.

If you have not already done so, load the BOOKRECD model into Javelin.

Editing Keys

■ No one types everything right the first time. In this section, you will learn how to fix typing mistakes by using the Edit key, as well as keys to move the cursor and to insert and delete characters in Edit mode.

■ The Edit Key

To change a formula or number, you can use the Edit key, F2, to copy the formula or number highlighted by the cursor to the edit line. The edit cue, a blinking underscore, appears on the edit line where what you type will appear. While information is on the edit line you are in Edit mode, and the EDIT indicator appears on the status line. You can make corrections using the ← (Backspace) key, the arrow keys, and other keys. When you press ↵ to indicate that the entry is complete, the information you have typed is entered into the model and appears in the view area and you return to Ready mode.

■ *Cue Movement, Insertion, and Deletion Keys*

When you are in Edit mode, you can use the editing keys listed in Table 5.1. Make some changes to your BOOKRECD model. Take a look at the formula for Book Expenses in the Formulas view; if no Formulas view is displayed, move your cursor to the bottom window and give the command **/ View Formulas All**. If you recall the discussion about top-down design in Chapter 1, you will notice that a formula like

Book Expenses = Book Volume * (0.75 * Book Price)

is bad modeling practice, for two reasons. First, nowhere is 0.75 identified as the markup on books, and second, nowhere is Book Cost identified. You can fix both of these things by adding the variables Book Cost and Book Markup to the model and redefining Book Expenses in terms of Book Cost and Book Volume. While doing so, you will make extensive use of the Edit key and the cursor-movement keys in Edit mode.

Move the cursor to the Book Expenses formula in the Formulas view. The part of this formula that says (0.75 * Book Price) represents the Book Cost. First, to rewrite the Book Expenses formula, press the Edit key, F2. The Book Expenses formula moves to the edit line at the top of the screen. The

Key Name	How Key is Used When Editing
←	Moves the cue one character to the left.
→	Moves the cue one character to the right.
Ctrl-←	Moves the cue one word to the left.
Ctrl-→	Moves the cue one word to the right.
Home	Moves the cue to the beginning of what you are editing.
End	Moves the cue to the end of what you are editing.
Del	Deletes the character the cue is on.
⟵	Deletes the character to the left of the cue.
Ctrl-⟵	Deletes the word to the left of the cue.
Esc	Cancels the edit and leaves the formula as it was before you started editing.

Table 5.1: Meanings of keys in Edit mode

edit cue is at the beginning of the formula. Now, position the cue on the 0 in 0.75 by pressing Ctrl-→ four times. This action moves the cue word by word through the edit line. Press the Del key seven times to delete *0.75 ** from this formula.

Delete the parenthesis to the left of the cue by pressing ←. Replace the word Price with Cost. Press End to move the cue to the end of the edit line. Delete the close parenthesis by pressing ←. Delete the word to the left of the cue (Price) by pressing Ctrl-←.

Complete the formula by typing

Cost

Your formula should read:

Book Expenses = Book Volume * Book Cost

When the formula on the edit line looks right, press ↵ to store this corrected formula as part of the model. The keys on the edit line can also help you type in the formula

Book Cost = Book Price * (1 - Book Markup)

Move the cursor to Book Price and press Edit. Take advantage of the fact that characters are inserted on the edit line as you type, and enter

Book Cost =

Notice that the formula for Book Price has been pushed to the right. Now move to the end of the line and erase = 7.5 by pressing End and Ctrl-← twice. Type the rest of the formula:

*** (1 - Book Markup)**

After you press ↵, your formula should read:

Book Cost = Book Price * (1 - Book Markup)

Finally, enter the Book Markup. Notice that Book Markup has appeared on the formulas list. Position the cursor on it and edit it by pressing Edit (F2). Type the value to mark the books up (.25 represents 25 percent):

= .25

The formula should read:

Book Markup = 0.25

The objective of this exercise was to do two things: to familiarize you with the Edit key and the keys that work on the edit line, and to clean up your BOOKRECD model a little. Whenever you need to change something that is on the screen, try using Edit and the cursor-movement keys that go with it. Eventually, they will come to be second nature and save you a lot of typing.

Save the model with all these changes by giving the command / **F**ile **S**ave **Y**es (that is, save it as BOOKRECD, replacing the previous version).

The Goto Key

■ In editing your list of formulas, you used ↑ and ↓ to move from variable to variable. A more direct way to move around the view you are in is to use F5, the Goto key. When you press F5, Javelin asks you on the edit line for the name of the variable you wish to go to. Type the variable name and press ⟵. If that variable is displayed in the current view, the cursor will appear on it. If this means that Javelin has to scroll the view (up, down, left, or right), it will do so. If the variable is not displayed in the current view, Javelin will beep. If the variable does not exist in your model, Javelin will beep and leave what you typed on the edit line. All the editing keys you just practiced with can be used to correct it.

The Goto key can also be used to go to a time period for a variable. If you are in a Table view looking at the values for one variable over many periods, you can go directly to a time period just by pressing F5 and typing the name of the period, for example, Dec 87 or 3Q87 or 1990. The same is true in a worksheet, where you look at both many variables and many time periods. We'll illustrate this in the examples that follow.

The Goto key is most useful in large views—those that display a large number of variables and/or time periods. Our Formulas view is so short that it is easy to move up or down the list with the ↑ and ↓ keys. But for practice, move to the Profit formula using the Goto key. First press F5, then type

Profit

When you press ⟵, the cursor moves.

Now use the Goto key to move around the Worksheet view. Switch views by pressing F6, the Window key, and view the Comparison worksheet (/ **V**iew **W**orksheet Comparison). To position yourself on Record Expenses, press F5, then type

Record Expenses

When you press ⟵, the cursor moves to the cell where the Record column and Expenses row intersect. To see what happens if you ask Javelin to go to Profit, press F5, and type

Profit

Javelin beeps: Profit is not in this view.

Try using the Goto key in another Worksheet view. Display the P&L worksheet by giving the command / **V**iew **W**orksheet P&L. Again, you are in the upper-left corner. Now try to see Profit (which isn't on the screen but is on the worksheet). Press F5, then type

Profit

The cursor appears on Profit for 1 Q 1987, on the top line of the view. Whenever you ask to go to something off the screen, Javelin places the cursor in the top-left corner of the screen.

■ *Going to a Time Period*

Let's display a Table view in the other window and see how you can go to a specific time period. First, to change windows, press F6. Then look at a Table view by giving the command / **V**iew **T**able. Type

Profit

Go to December 1987 by pressing F5 and typing

Dec 87

The time period you want may be larger or smaller than the time period of the variable you are looking at. Try the following: press F5 and type

2Q87

Try going to 1987, or 15 Aug 87. In each case, Javelin goes to the first period displayed that contains the date you typed.

Finally, combine the variable and date aspects of the Goto key. The @ ("at" sign) means you want to see the variable at this date. Press F5 and type

Book Expenses @ 4Q87

The Table view changes to display Book Expenses and the cursor highlights October 1987, the first month of the fourth quarter.

You will probably find the Goto key most useful in large worksheets, long lists of formulas, or any Diagram view (described in Chapter 6, "Javelin Views and Windows").

The Spell Key

■ Because Javelin lets you use long, descriptive names for your variables, it can be time consuming to type the variable names when you create formulas or give commands. To speed up this process and to remind yourself what variables you have already defined, you can use the Spell key, which is the ⟶| key on the keyboard.

When you type in a formula or give a command, and you want to enter the name of a variable, you can use the Spell key to ask Javelin for help. Start typing the variable name and then press the Spell key. If only one variable matches what you have typed so far, Javelin completes the name. If there is more than one possibility, Javelin displays a *possibilities menu,* a list of the possible names with the cursor on the first one. You can move the cursor from choice to choice with ↑ and ↓. Press ⟵ when the cursor highlights the variable name you want (see Figure 5.1).

In fact, you can use the Spell key to show your choices any time you are asked for a name—such as a variable name, a file name, a worksheet name, or a graph name.

When you have pressed the Spell key and the menu of possibilities is on the screen, you can use the keys ↓, ↑, Home, End, PgDn, and PgUp to move up and down in the list.

Try using the Spell key in conjunction with the Goto key. Right now, you should be looking at a Table view of Book Expenses. To see what other variables relate to Books, press F5 and type

Book

Next, press ⟶|. A possibilities menu appears. Choose the variable you want by highlighting it and pressing ⟵┘. Select Book Sales.

Find what worksheets are available to look at using the command **/ View Worksheet** and pressing ⟶|. A possibilities menu appears, listing the Comparison worksheet, the P&L worksheet, or a blank worksheet called Worksheet. Select Comparison.

■ *The Spell Key and Word Order*

There is only one limitation to the Spell key: it shows you the words in the names of your variables, worksheets, and other parts of the model in the same order as you typed them. For example, typing either of these will get you to Book Expenses (preceded by pressing F5):

Expenses Book
Book Expenses

as will typing

Book

pressing ⟶| and selecting Book Expenses. But if, after F5, you type

Expenses

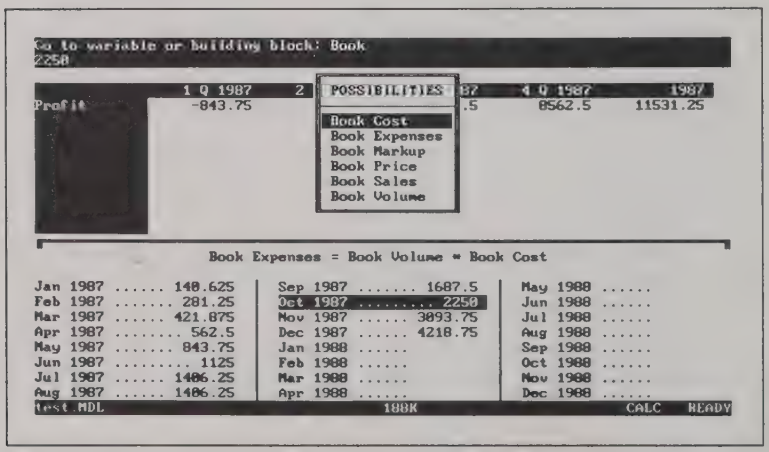


Figure 5.1: Possibilities menu displayed by the Spell key

and press ⟶|, Javelin will beep, telling you that there are no variables beginning with the word Expenses. This is true. Although Javelin lets *you* specify the words making up a variable name in any order, *it* remembers them using the same word order you typed them in originally. (If you find you have made a mistake and created one variable as Book Expenses and another as Expenses Record, you may change one of the names using / **D**efine **V**ariable, described in Chapter 8, "Formulas and Functions".)

Spell presents options in alphabetical order (except for files, which are listed in the order they appear on the disk). You will find the Spell key very useful throughout Javelin whenever you have to specify the name of something.

Settings Sheets

■ When you typed in the formulas for the variables in Chapter 3, Javelin created the variables for you. But there is a great deal of detail information (called *settings* in Javelin) that affects variables, such as the format the number should be displayed in and the number of decimal places it should show. You can look at and change this detail information by using a *settings sheet*.

A settings sheet is a full-screen display of all settings that are in effect for a particular part of a model. There are settings sheets for variables, for worksheets, or for the entire model (see the list of settings sheets in Table 5.2). All settings sheets work the same way. In Javelin PLUS, additional settings have been added to many of the settings sheets. To introduce settings sheets, we will look at three of them: the Model settings sheet, the Configuration settings sheet, and (briefly) the Variable settings sheet.

■ Using Settings Sheets

Let's begin by looking at the Model settings sheet, which controls the time span for the model, the periodicity for most of the variables (daily, monthly, and so on), how numbers should be displayed, and other aspects of the model. To see the Model settings sheet, use the command / **D**efine **S**ettings.

The settings sheet appears and covers up most of the screen. Figure 5.2 shows how the Model settings sheet appears in Javelin PLUS; in Javelin 1.1,

not all of these settings appear. The Model settings sheet is divided into several areas: Name, Default Time Span, Default Period, Fiscal Year, Decimal Places, Format, End Day of Week, and Splitting Method. In each area, one setting is in effect. The cursor starts on the Name setting, which is highlighted. You can move the cursor from setting to setting using the arrow keys, ←, →, ↑, and ↓. The settings on the Model settings sheet apply to all variables in the model, although you can override them for individual variables by using the command / **D**efine **V**ariable.

When you have finished making changes on a settings sheet, press Esc to return to Ready mode. This saves the settings as you have set them. Any views that were on the screen when you asked to see the settings sheet are still there.

There are three types of settings: value settings, multiple-choice settings, and toggle settings. You specify the three kinds of settings differently. Here is how to use each.

Name of Settings Sheet	Command to Display the Settings Sheet	Chapter Where Described
Variable	/ **D**efine **V**ariable	8, "Formulas and Functions"
Worksheet	/ **D**efine **W**orksheet	7, "Defining Variables and Entering Data Using Worksheets"
Graph	/ **D**efine **G**raph	12, "Graphing"
Custom Period	/ **D**efine **T**ime period	10, "Custom Time Periods"
Printer	/ **D**efine **P**rinter	13, "Advanced Worksheets and Printing"
Configuration	/ **D**efine **C**onfiguration	5, "Special Keys and Settings Sheets"
Model	/ **D**efine **S**ettings	5, "Special Keys and Settings Sheets"
Format	/ ***W**orksheet **F**ormat	13, "Advanced Worksheets and Printing"
Number	/ **D**efine **N**umber	8, "Formulas and Functions"

Note: The Number settings sheet does not exist in Javelin version 1.0.

Table 5.2: List of settings sheets

■ *Value Settings*

These are settings that can be set to a number, date, or text you type in. On the settings sheet, the name of the setting is followed by the value of the setting in braces: { }. If the setting has no value yet, there may not be anything between the braces.

Display the Model settings sheet again, by giving the command / **D**efine **S**ettings. The Name setting is an example of a value setting, as is the Decimal Places setting (see Figure 5.2).

Set the decimal places at two (for dollars and cents). This is the appropriate format for most of the variables in the model (it can be overridden for a specific variable using the Variable settings sheet for that variable).

Move to the Decimal Places setting by pressing →, and type

```
2
```

When you start to type in the value, a prompt appears on the edit line to remind you what you are entering. Then press ⟵, which enters the value for the setting onto the settings sheet between the braces. The setting looks like this: {2}.

A value setting can be edited using the Edit key, F2. Change the value and press ⟵ to store the change on the settings sheet.

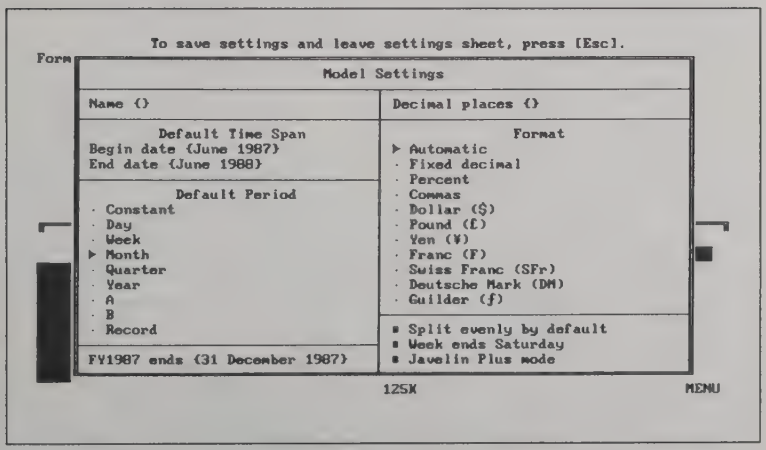


Figure 5.2: The Model settings sheet in Javelin PLUS

■ *Multiple-Choice Settings*

In these settings a number of choices are listed, of which you pick one. Each choice in the list is preceded by a dot. The choice that is in effect is preceded by a right-pointing triangle.

The Format setting is an example of a multiple-choice setting (see Figure 5.2). The setting currently in effect is Fixed decimal, which means that all numbers will be shown with the same number of places after the decimal point, according to the Decimal places setting. Set the format to Dollar (this displays a dollar sign before the number and commas separating each group of three digits to the left of the decimal point).

Press ↓ five times, then ⟵. This deselects Fixed decimal and selects Dollar for the Format setting, and the triangle moves to the left of Dollar. Exit from the settings sheet by pressing Esc. The new settings go into effect, changing the way in which numbers are displayed.

■ *Toggle Settings*

There is one kind of setting not found on the Model settings sheet: toggle settings. Toggle settings are like multiple-choice settings, but you see only one choice at a time. Each setting is preceded by a box. Let's use the Variable settings sheet for an example of toggle settings. (See Figure 5.3.)(This is described fully in Chapter 8, "Formulas and Functions.") To display a Variable settings sheet, give the command / **D**efine **V**ariable, then type

Book Expenses

The only toggle settings on the Variable settings sheet are the two Definition settings: Locked/Unlocked and Fill with NA's/Fill with blanks (fill with blanks is fill with 0's in Javelin 1.1). You can set a variable to Locked to prevent it from being accidentally changed or deleted. Fill with NA's/Fill with blanks tells Javelin how to deal with missing values for the variable. Both of these settings are described fully in Chapter 8. For now, we'll use them just to illustrate the use of toggle settings.

Tell Javelin to lock this variable so the formula cannot be changed. Move to Unlocked by pressing →, then pressing ↓ ten times. Change this to Locked by pressing ⟵. The setting changes to Locked. Change it back by again pressing ⟵. It switches back to Unlocked.

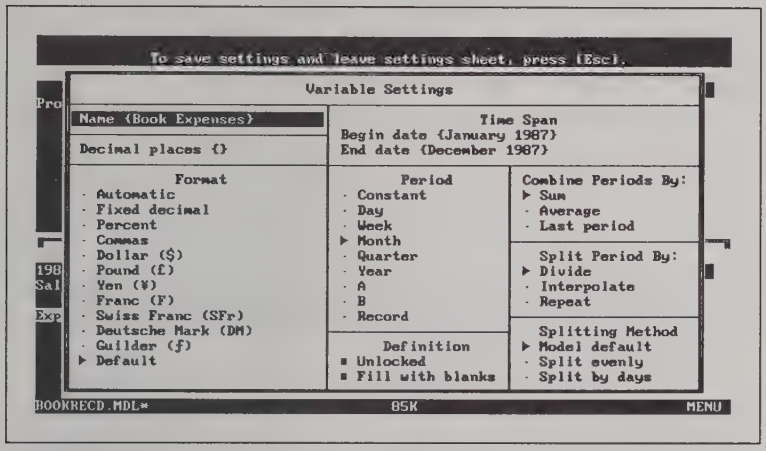


Figure 5.3: The Variable settings sheet in Javelin PLUS

Some toggles have more than two possible settings. You press ⟵ to flip from one to the next. In Javelin PLUS, you can also press the Spell key to see a menu of all the possible settings. Pressing Del returns the toggle to its original default setting.

Exit from the settings sheet using Esc.

The Model and Configuration Settings Sheets

■ Settings sheets specify many characteristics of variables, graphs, and models. In general, we will describe them in the chapters on their subjects (worksheets, graphing, and others). However, there are two settings sheets that determine a great deal about how Javelin behaves, so we will describe them here. They are the Model settings sheet, which you used above, and the Configuration settings sheet.

Table 5.3 contains descriptions of each setting on the Model settings sheet (shown in Figure 5.2). These settings apply to all variables in the model.

Many settings can be overridden for a specific variable by changing the setting on the Variable settings sheet. If a setting is left blank, the Model setting applies. Some of these settings are new in Javelin PLUS.

Setting	Description
Name	File name of the model; if the model has not yet been saved, this is blank. Changing the name of the model here changes the name that will be used when you give a / **F**ile **S**ave command. You may use a complete path name here, but you should probably use the Directory setting on the Configuration settings sheet instead (see Table 5.4). You can change the name of the model by editing this setting.
Default Time Span	Span of time over which the model runs. If you do not change it, your model is assumed to start in the month the model was created and to end one year later. *This is the single most common modeling hangup in Javelin!* If you find your formulas behaving peculiarly, or your worksheets giving you the error #NODATE, this is probably the problem. When you are setting up your model, always set the default time span to something appropriate given your data. When in doubt, make the default time span longer than you need to. See Chapter 9, "Javelin and Time," for more information.
Default Period	Periodicity of variables, i.e., daily, weekly, monthly, quarterly, yearly, or custom time periods. If you don't change it, the default is monthly; this can be overridden for specific variables. See Chapter 9 for more information.
FY	Last date of the fiscal year. Change this only for years that do not end in December; this is described fully in Chapter 9.
Decimal Places	Number of decimal places to use when displaying numbers. This can be overridden for specific variables by using the Variable settings sheet (see Chapter 8, "Formulas and Functions") and on worksheets (see Chapter 7, "Defining Variables and Entering Data Using Worksheets").
Format	Format to use when displaying numbers. As with Decimal Places, this can be overridden for specific variables and within worksheets. For full information, see Chapter 8.
Splitting Method	How periods are split or combined when Javelin must convert a variable from one period to another. See Chapter 9.
Week Ends	(Javelin PLUS.) Day of the week Javelin should use or the last day of each week. The default is Saturday, so weeks start on Sunday and end on Saturday.
Calc Mode	Determines whether to calculate by Javelin 1.1 or Javelin PLUS rules.

Table 5.3: Settings on the Model settings sheet

Figure 5.4 shows the Configuration settings sheet, which contains general information about your computer hardware, where Javelin should look for your models, and the style in which you wish to view your numbers. Table 5.4 describes each setting (some settings exist only in Javelin PLUS). The command to display the Configuration settings sheet is / **D**efine **C**onfiguration. All the settings that affect how numbers are displayed are explained in Chapter 8, "Formulas and Functions."

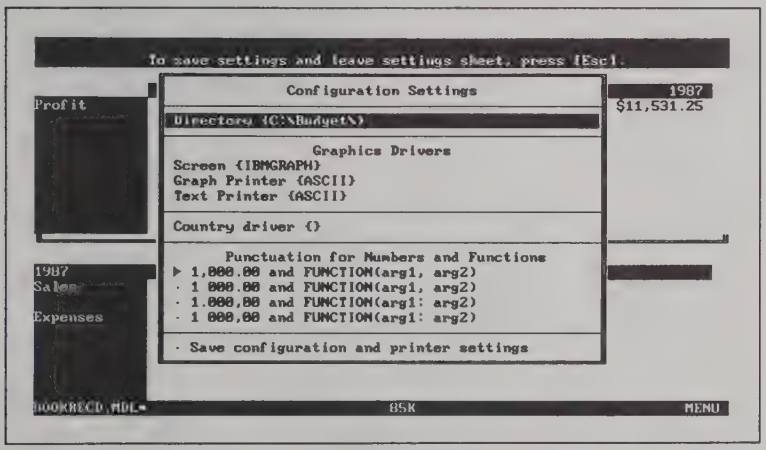


Figure 5.4: The Configuration settings sheet in Javelin PLUS

Setting	Description
Directory	The path name of the directory in which Javelin will look for your models (path names are described above in the section on file operations). If you store your models in more than one directory, you will want to change this setting depending on which models you are working on. You can use the Spell key to display all possible subdirectories. For instance, type C:\ and press ⟶| to list all subdirectories in the root of drive C.
Screen	The name of the "driver" that Javelin should use to display graphics on your screen. This depends on your hardware configuration. As usual, you may use the Spell key to look at your options—in this case, all files ending

Table 5.4: Settings on the Configuration settings sheet

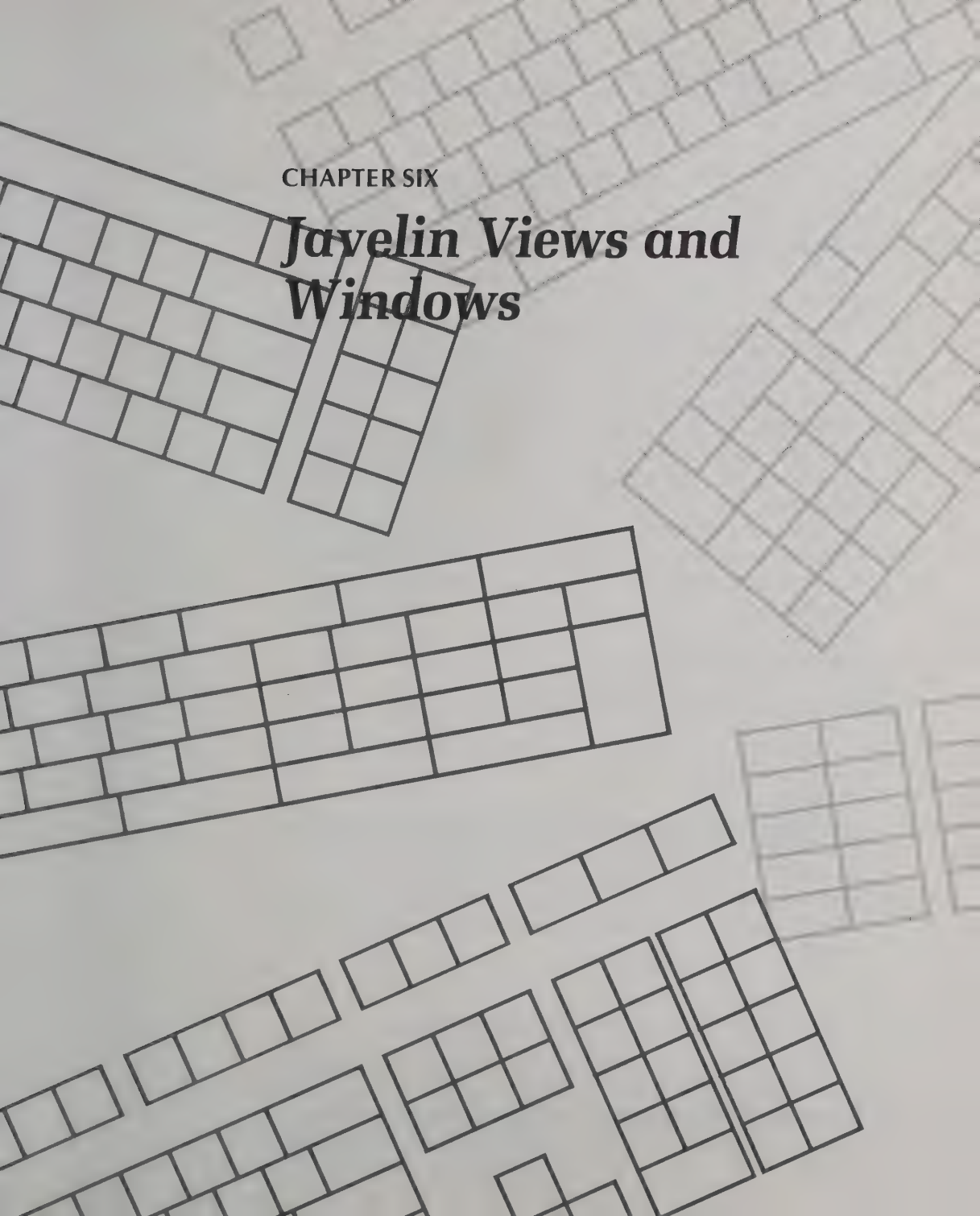
Setting	Description
	with .DRV in the \JAV subdirectory. You should only need to change this the first time you use Javelin, *provided you save your configuration settings* (see below).
Graph Printer Text Printer	Similar to Screen above, these are the names of the "driver" Javelin should use to print graphs and text. Change them only when you first use Javelin or if you get a new printer.
Country Driver	(Version 1.1 and later.) Selects the words Javelin uses for month names, spelled-out names of numbers, function names, etc. Normally, these are all in English, but international users can select the language of their choice.
Decimal Point and Comma	(Version 1.0 only.) Selects U.S. punctuation for numbers (for example, 1,234.56) or European punctuation (for example, 1.234,56). Numeric formats are described in Chapter 8, "Formulas and Functions."
Punctuation for Numbers and Functions	(Version 1.1 and later.) Allows you to choose among four styles of punctuation for *all* numbers (regardless of other formatting considerations). This setting determines whether the decimal point is a period or a comma, and whether thousands, millions, etc., will be set off by commas, periods, or spaces. If you choose to use a comma as the decimal point, function arguments are separated by colons rather than commas. For more about functions and numeric formats, see Chapter 8.
Save Configuration Settings	Records the current configuration settings so that the next time you use Javelin, they appear as they are now. If you have changed the drivers or the numeric format, you probably want to save the settings.

Table 5.4: Settings on the Configuration settings sheet (continued)

Conclusion

■ In this chapter, we have explained some of the more useful keys in Javelin: F2 (Edit), F5 (Goto), and ⟶| (Spell). We also introduced settings sheets, used throughout Javelin to specify details about variables, graphs, and other elements of your models.

You should now have a good grasp of *how* to make Javelin do things. In the next chapter we'll show *what kinds* of ways Javelin shows your results.



CHAPTER SIX

Javelin Views and Windows

When you create a model in Javelin, the formulas, data, settings, and other information are stored in a *central information base* for the model, which you look at through views. Depending on your view, the model will look different, but the information is the same. If you make a change to your model in one view, the changes are reflected in all other views. You can also use windows to see two views at the same time.

In this chapter we will look at each of the ten Javelin views in more detail, so you can see how each is useful. We will also look at the commands that let you control the number and size of the windows on the screen. The main commands you will use are / **W**indow and / **V**iew.

In addition, Javelin has commands that are available only in a particular view. When your cursor is in certain views, the Javelin command menu shows an additional choice, the rightmost choice on the menu. The name of the additional menu choice is the name of the view preceded by an asterisk. For example, commands that are available only in the Worksheet view are called / *Worksheet commands. These special commands are also known as *star commands,* and you will learn how to use them in this chapter.

Controlling Windows: The / Window Command

■ Javelin can display either one or two views of your model at one time. You control the number and size of the windows using the command / **W**indow.

Start up Javelin but do not load a model. When Javelin first loads, the screen is divided into two windows, with a Diagram view in the top window and a Table view in the bottom window. In Javelin PLUS, the top window shows a Formulas View and the bottom window shows a Worksheet view. The window divider is halfway down the screen, making two windows of equal size.

■ *Moving the Cursor Between Windows*

As you saw in previous chapters, you can use F6, the Window key, to switch the cursor from one window to the other. Another way to do this is with the / **W**indow **S**witch command. There is no good reason to choose one of these ways of switching windows over the other—the effect is the same.

■ Moving the Window Divider

You may want to make one window larger than the other—for example, a large Worksheet window for creating a report with a small Formulas window for reference.

You can move the window divider up and down the screen using the / **W**indow **A**djust command. When you give this command, Javelin lets you move the window divider up and down the screen using ↑ and ↓. When the divider is where you want it, press ⟵ to complete the command (see Figure 6.1).

Try it now—make the bottom window smaller. Give the command / **W**indow **A**djust, then move the window divider three lines down the screen, using ↓. To finish the command, press ⟵.

■ Swapping Two Windows

If you decide you'd like the bottom window on top and vice versa, you can exchange the two windows (this command doesn't exist in Javelin 1.0). To swap the two windows, give the command / **W**indow **E**xchange.

The contents of the windows switch, the top moving to the bottom and the bottom moving to the top. The windows themselves stay the same size. A shorter way to do the same thing is Alt-F6, that is, hold down the Alt key and press F6.

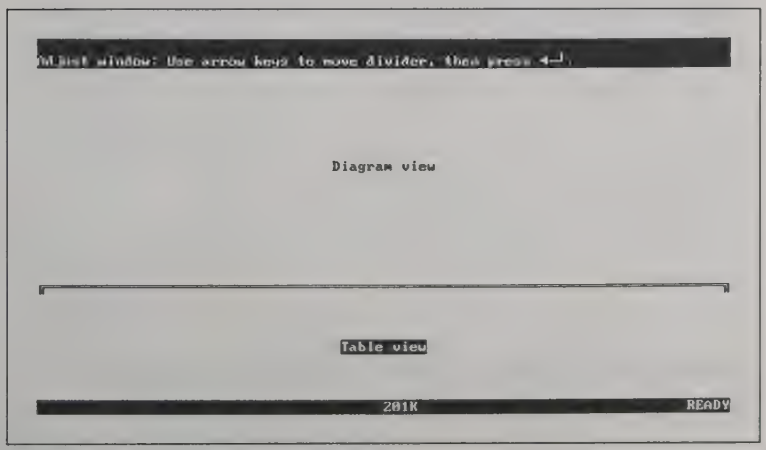


Figure 6.1: Adjusting the sizes of windows

■ *Choosing One or Two Windows*

At times, you may want to see only one window on the screen, most often when you are developing a large worksheet, although it can also be useful in the Formulas view. To tell Javelin to display one window, use the command / **W**indow **1**-window. The window the cursor is in expands to take up the whole screen, and the other window disappears.

If you haven't done so already, make one window take up the whole screen. Give the command / **W**indow **1**-window; to see two windows again, use / **W**indow **2**-windows.

The view is scaled back to half the screen, and another window appears containing exactly the same view. When you go from one window to two, the bottom window always becomes the active one. This is useful if you are writing macros (see Chapter 14, "Macros"). Notice that you lose any changes you may have made to the sizes of the windows. When you go from one window to two windows, you get two windows of equal size.

If Javelin is displaying one window, it ignores the / **W**indow **1**-window command, and if it is displaying two windows, it ignores / **W**indow **2**-windows. There is no way to see two different models in the two windows—they are always two views of the same model.

Looking in Windows: The / View Command

■ To choose the view you want to see, use the command / **V**iew. If your screen is showing one window, the view you select will appear on the whole screen. If the screen shows two windows, the view you select will appear in the window your cursor is in when you give the command. Remember, you can tell which window your cursor is in by checking the ends of the bar that separates the windows.

Now load the BOOKRECD model you created by giving the command / **F**ile **L**oad BOOKRECD. The rest of the examples in this chapter will refer to this model.

Make sure that the cursor is in the bottom window (press F6 if it is not). Give the command / **V**iew to display a view of the model, and the View menu appears (see Figure 6.2—your screen may show different views).

This is the list of views available to you; they are described in Table 6.1. Each view is valuable for certain tasks. We will look in depth at six of the

views you can use for creating and manipulating the model: the Diagram, Formulas, Table, Chart, Quick-graph, and Notes views.

The other four views will be reviewed in more detail later; for now we'll just look at them briefly. In the next chapter, you will see how to use the Worksheet view for creating and modifying your model. In Chapter 11, "Special Values and Diagnosing Problems," we will explain the kinds of problems you may encounter and how to use the Errors view to solve them. In Chapter 12, "Graphing," we'll cover how to create and print graphs with the Graph view. And in Chapter 14, "Macros," you will learn to use macros and the Macro view.

The Diagram View

■ In the Diagram view, Javelin presents a road map of your model. In the center of the screen is the variable you are currently looking at. To the left are the variables that feed information into the current variable. To the right are variables that take information from the current variable. Think of the diagram as a tree turned on its side: like your data, water and nutrients flow from roots on the left to branches on the right. That is why the variables on the left are called the roots of the current variable, and the variable on the

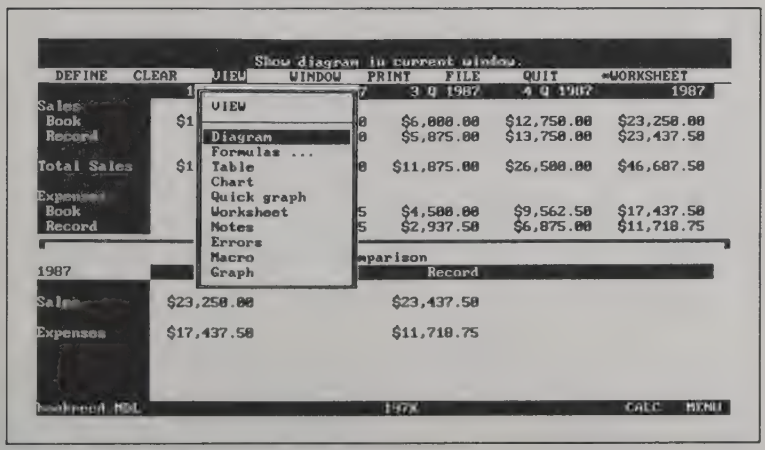


Figure 6.2: The View menu

right are called the branches of the current variable. Also like a tree, branches may grow from branches and roots may grow from roots.

A variable that has no roots is called an *input variable*—one for which you enter the values directly, rather than calculating them from other variables. Input variables are like the smallest roots of the tree. A variable that has no branches is a *result variable*—a final result of the model, and no other variables are calculated based on it. Result variables are like the leaves of the tree.

To see a Diagram view, use the command / **V**iew **D**iagram. Choose a Diagram view of the Book Sales variable. You should still have the View menu on the screen, so select **D**iagram from it. Javelin prompts you for the name of the variable to be displayed in the middle of the view. Press ⟶|, the Spell key and select Book Sales.

View	Description	Star Commands
Diagram	Shows how variables interrelate and lets you create and edit variables and formulas.	None
Formulas	Shows formulas for each variable and lets you create and edit variables and formulas.	/ *Formulas, / *Roots, and / *Branches
Table	Shows values for one variable in a table and lets you enter and edit values.	None
Chart	Shows values for one variable as a bar chart and lets you enter and edit values.	None
Quick-graph	Shows values for one variable as a line graph and lets you enter and edit values.	None
Worksheet	Shows data as a worksheet and lets you enter and edit variables, formulas, and data as well as creating reports.	/ *Worksheet
Notes	Shows comments about one variable and lets you enter and edit comments.	None
Errors	Shows list of errors in the model with explanations.	/ *Errors
Macro	Shows macros you have written and lets you enter and edit them.	None
Graph	Shows graphs you have created.	None

Table 6.1: The ten Javelin views

The Book Sales variable is shown as a box in the middle of the window. The variables on the left side of the screen are the roots of Book Sales: Book Volume and Book Price. The variables on the right side of the screen are the branches of Book Sales, in this case Total Sales. Arrows show the flow of data from left to right. At the bottom of the view is the formula for the variable that your cursor is on (see Figure 6.3).

You can move your cursor from box to box, but don't do so yet—it's easy to get lost in the Diagram view. Just like other Javelin views, the Diagram view will scroll (in this case to the left or right) if you try to move the cursor off the edge of the screen, and other variables will come onto the screen. You can move from variable to variable, working either "up" the tree to the right or "down" the tree to the left. But watch very carefully what happens as you move.

Move one variable to the right, using →. As you would expect, the cursor moves to Total Sales and the formula for Total Sales appears at the bottom of the window. You still see Book Sales feeding into Total Sales. Now move to the right again.

Javelin scrolls to the left so that Total Sales is in the middle of the screen. The variables affected by Total Sales appear on the right (in this case Profit and Shipping Expenses). But on the left you find that not only is Total Sales affected by Book Sales, it is also affected by Record Sales. This is not as surprising as it might seem: when Total Sales was on the right edge of the screen, you were looking up the tree from the point of view of Book Sales. While

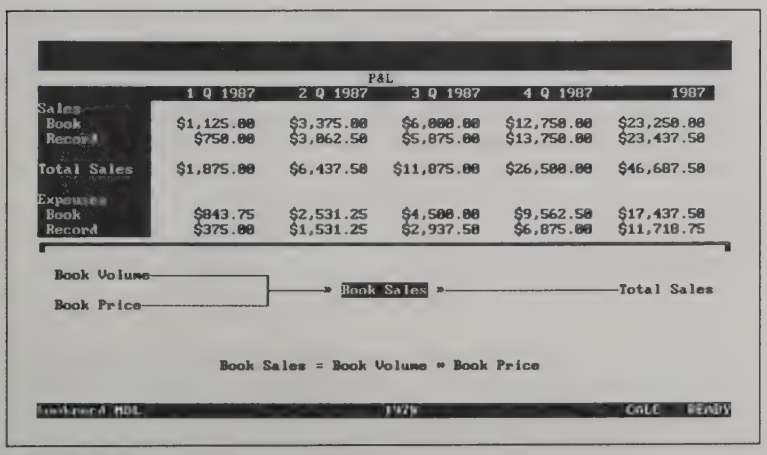


Figure 6.3: Diagram view of Book Sales

Javelin might have provided on the diagram some indication that another variable also influences Total Sales, a quick glance at the formula (displayed at the bottom of the window) makes that clear. Also, moving right again brought Total Sales into the middle of the screen, where all its roots became apparent on the diagram.

Now press ←. Javelin moves the cursor to Book Sales. Just as when Total Sales was on the right edge of the screen, not all possible branches coming from Book Sales are shown, only the branch leading to Total Sales, since this is the point at which you are looking down the tree. Why did Javelin move the cursor to Book Sales and not Record Sales? Because it remembered that you went to Total Sales from Book Sales. To look at Record Sales, just press ↓ and Record Sales is highlighted, with its formula displayed at the bottom of the window. If you press ←, Record Sales moves to the middle of the screen and its roots (Record Volume and Record Price) appear on the left side of the screen. To return to Book Sales, press →, which moves you to Total Sales on the right edge of the screen and does not show all its roots.

Pressing → again moves Total Sales to the middle of the screen and shows you both of its roots: Book Sales and Record Sales. Now press ←. The cursor moves back onto Record Sales since that is the path you took most recently to get to Total Sales. Press ↑ to return to Book Sales, then press ←, and Book Sales returns to the middle of the screen, with its roots on the left and its branches on the right.

You've probably noticed the following points. First, Javelin shows only the roots and branches of the variable in the center of the screen. This is obvious if the cursor is in the center of the screen, but less obvious if the cursor is on the right or left side of the screen. Even with the cursor on a side of the screen, the view is still of the roots and branches of the center variable. The only thing that changes is the formula displayed at the bottom of the window.

Second, traveling around the tree is a three-step process. Starting from the center of the screen, you indicate first whether you wish to go up or down the tree by moving right or left. Next, you indicate which branch you wish to take along the tree by moving up and down through the list of branches or roots. Finally, you bring the selected variable to the center of the screen by pressing ← if you are going down the tree or → if you are going up the tree.

But there is a more direct way to move the Diagram view to another variable. Use the Goto key, F5, to move the cursor directly to the Profit variable. Javelin asks which variable you want to go to. Type

Profit

The view changes to show Profit in the center. Note that no variables are affected by Profit because Profit is the end of the line.

■ Printing from the Diagram View

You can use the command / **P**rint **C**urrent view to print a Diagram view, but this can be tricky if your printer doesn't print graphics characters. If you get strange characters when you try to print out a diagram, tell Javelin to substitute nongraphics characters that your printer can print. Display the Printer settings sheet using the command / **D**efine **P**rinter and select ASCII Character Printer. For complete information on the Printer settings sheet, see Chapter 13, "Advanced Worksheets and Printing." There is no way to print the entire Diagram view; Javelin will only print the part of the Diagram view that is on the screen.

The Formulas View

■ One of the best ways to get an overview of your model is to see a list of the variables and their formulas—the Formulas view. For input variables, which have no formula, Javelin indicates the period and time span of the data, such as *Monthly data January 1987–December 1987.*

When you select the Formulas view, you must select what formulas you wish to view. You can list all the variables in the model (in alphabetical order), you can list only those variables that affect a certain variable (the roots of that variable), or you can list only variables affected by a particular variable (the branches coming from that variable). In addition, using the command / *Formulas, you can select from among a number of *categories* of variables to look at.

Switch to the top window by pressing F6, then give the command to display a Formulas view: / **V**iew **F**ormulas. An options menu appears (see Figure 6.4).

■ Viewing All Formulas

From this menu, select **A**ll to get a list of formulas for all variables in the model in alphabetical order (see Figure 6.5). As with all lists in Javelin, you may move around this list of formulas by using the ↑ and ↓ keys; the PgUp

and PgDn keys will show the previous or next screens of variable names. In addition, you may use F5, the Goto key, to go directly to a variable. (As usual, the Goto key is most useful in conjunction with ⟶|, the Spell key; pick the variable you wish to go to from the list that Spell presents.)

When your cursor is in the Formulas view showing all formulas, there is a set of star commands available, the / *Formulas commands. Press / and the rightmost command on the Main menu is now *Formulas. Select it by

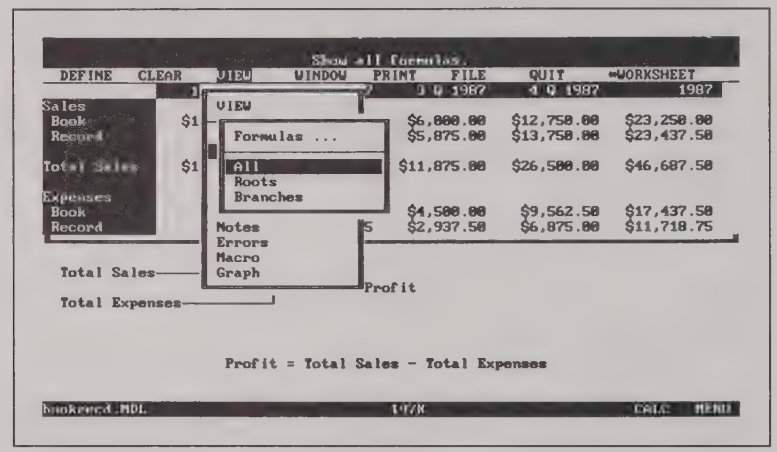


Figure 6.4: The Formulas options menu

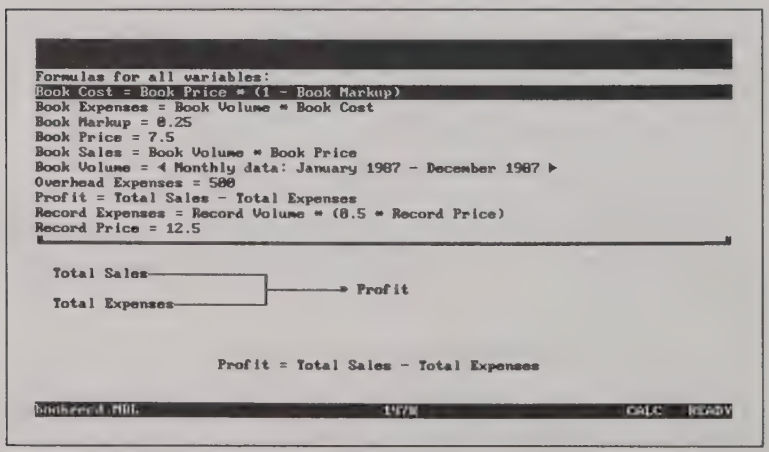

Formulas for all variables:
Book Cost = Book Price * (1 - Book Markup)
Book Expenses = Book Volume * Book Cost
Book Markup = 0.25
Book Price = 7.5
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead Expenses = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
Total Sales
Total Expenses
Profit
Profit = Total Sales - Total Expenses

Figure 6.5: The Formulas view: all formulas

moving to it or by pressing *, and you will see the *Formulas options menu, which is also called the *Formulas star menu* (Figure 6.6).

These commands let you display formulas just for input variables, just for final result variables, just for undefined variables, and for others. The Formulas options menu commands are listed in Table 6.2.

Let's see only the input variables: select **I**nputs. These are the variables for which you have entered data, rather than defining them by formulas.

Give the command / *Formulas **R**esults to see the result variables. Profit is the only variable that is a final result: it is not mentioned in any other formula in the model.

■ *Viewing the Roots of a Variable*

If you are developing a model and want to see all the variables that affect a certain variable, you can view a list of the roots of that variable. The list of variables appears with their formulas, in the order in which they affect the variable you specify.

Ask for the roots of the Total Sales variable, giving the command / **V**iew **F**ormulas **R**oots. Then press ⟶| and select Total Sales.

Figure 6.7 shows the Formulas view you will see. These are the variables that appear in the Diagram view to the left of the variable you specified. The formula for the variable you specified is listed first, and formulas for the variables that affect it are listed below it. The variables that affect it

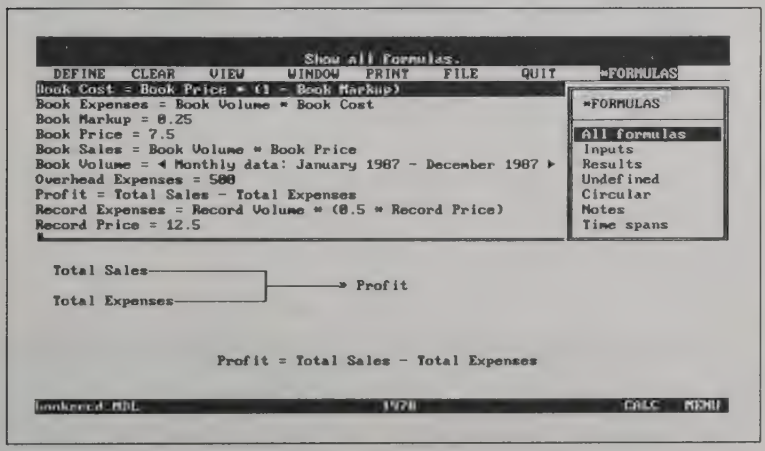

Show all formulas.
DEFINE CLEAR VIEW WINDOW PRINT FILE QUIT *FORMULAS
Book Cost = Book Price * (1 - Book Markup)
Book Expenses = Book Volume * Book Cost
Book Markup = 0.25
Book Price = 7.5
Book Sales = Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Overhead Expenses = 500
Profit = Total Sales - Total Expenses
Record Expenses = Record Volume * (0.5 * Record Price)
Record Price = 12.5
*FORMULAS
All formulas
Inputs
Results
Undefined
Circular
Notes
Time spans
Total Sales
Total Expenses
Profit
Profit = Total Sales - Total Expenses

Figure 6.6: / *Formulas options menu

Command	Description
All formulas	Shows all formulas (useful if you have selected one of the following commands, and want to go back to seeing all formulas).
Input	Shows only input variables rather than variables with formulas.
Result	Shows only variables that do not affect other variables.
Undefined	Shows only variables that depend on a variable whose value is undefined, or for which data have not yet been typed in.
Circular	Shows only variables that depend on themselves (perhaps with variables intervening—see Chapter 11, "Special Values and Diagnosing Problems").
Notes	Shows only variables that have notes entered for them using the Notes view.
Time spans	Instead of showing the formulas for variables, shows the time span for which each variable is defined (see Chapter 9, "Javelin and Time").

Table 6.2: / * commands in the Formulas view

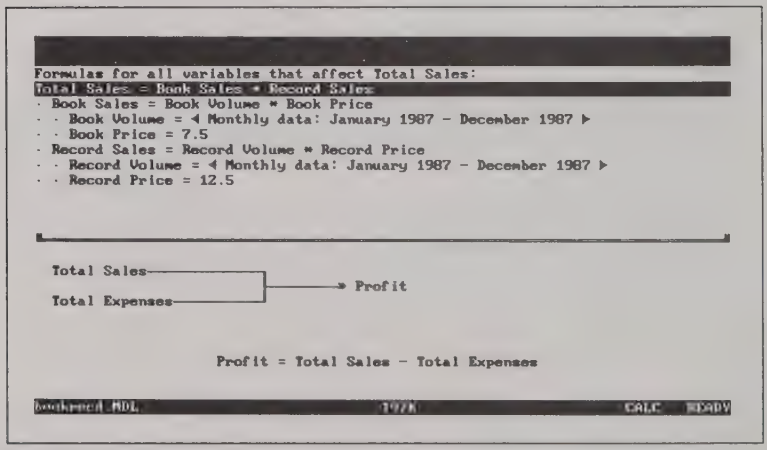

Formulas for all variables that affect Total Sales:
Total Sales = Book Sales + Record Sales
· Book Sales = Book Volume * Book Price
· · Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
· · Book Price = 7.5
· Record Sales = Record Volume * Record Price
· · Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
· · Record Price = 12.5
Total Sales
Total Expenses
Profit
Profit = Total Sales - Total Expenses
CALC
READY

Figure 6.7: The Formulas roots view

directly (shown in the Diagram view directly to the left of the variable) are indented and preceded by a dot. The roots of those variables are also shown, indented and preceded by two dots. Javelin continues down the roots of the variable until it gets to input variables, which have no roots. When one variable is a root of several others, it is listed under the first variable of which it is a root (in Javelin 1.1, it is listed under each variable of which it is a root).

The Formulas view showing roots is very useful when you want to see the details of how Javelin arrived at the values for a particular variable—you can trace the logic backward variable by variable.

F5, the Goto key, is particularly useful in the Formulas roots view. When you gave the command / **F**ormulas **R**oots, you told Javelin what variable you wanted to see the roots of (in the example above, Total Sales). The Goto key allows you to look at the roots of a different variable, even if it is not a root of the variable you selected. For example, Total Expenses is not a root of Total Sales. However, to select roots of Total Expenses you can use F5 and ⟶|. Select Total Expenses from the variables list.

When your cursor is in a window showing a Formulas view for the roots of a variable, there are special / *Roots commands available. The / *Roots menu is also called the *Roots star menu.*

These same commands appeared when you selected / **V**iew **F**ormulas **A**ll. (The choices Results and Circular are left off; they don't mean anything if you are just looking a variable's roots.)

To look at the roots of the Total Expenses variable that are input variables, give the command / *Roots **I**nputs. If you want to find out what input values affect Total Expenses, this is the way to do it.

■ *Viewing the Branches of a Variable*

You can also see a list of the branches of a variable. Starting with the variable you specify, Javelin shows which variables depend on it, which variables depend on those variables, and so on until it gets to the result variables, on which no other variables depend. To view the branches of the Book Volume variable, give the command / **V**iew **F**ormulas **B**ranches, press ⟶|, and then select Book Volume.

Figure 6.8 shows the Formulas view you will see. The variables here are those that appear in the Diagram view to the right of Book Volume. The formula for Book Volume is listed first, with formulas for the variables that are

affected by it listed below. The variables that are directly affected by it (shown in the Diagram view directly to the right of the variable) are indented and preceded by a dot. The branches of those variables are shown, indented and preceded by two dots. As in the Roots view, this continues until Javelin gets to result variables, which have no branches. When one variable is a branch of several other variables, it is listed several times. The Formulas view showing branches is useful when you are thinking of changing a variable and want to determine its impact on the rest of the model.

As with / **V**iew **F**ormulas **R**oots, if you wish to see the branches of some other variable (even one not currently on the screen, like Record Volume) you may use the Goto and Spell keys.

When your cursor is in a window showing a Formulas branches view, special / *Branches commands are available. The / *Branches menu is also called the *Branches star menu.* These are the same commands that appeared when you selected / **V**iew **F**ormulas **A**ll and / **V**iew **F**ormulas **R**oots. (The choices Roots, Circular, and Undefined are omitted because they don't mean anything if you are just looking at a variable's branches.)

To look at the branches of the Book Volume variable—those variables that will change if you change the value of Book Volume—give the command / *Branches **R**esults. If you want to know what result values will change if you change Book Volume, this is the way to do it.

Figure 6.9 shows the three different star menus in the Formulas view: the Formulas star menu, the Roots star menu, and the Branches star menu.

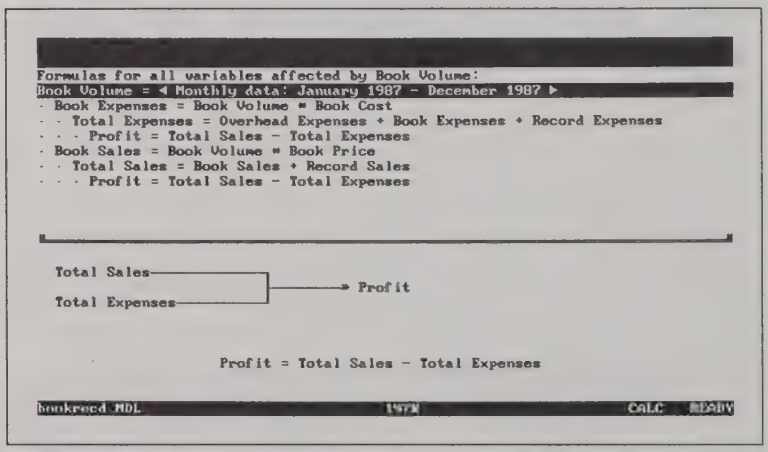

Formulas for all variables affected by Book Volume:
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
· Book Expenses = Book Volume * Book Cost
· · Total Expenses = Overhead Expenses + Book Expenses + Record Expenses
· · · Profit = Total Sales - Total Expenses
· Book Sales = Book Volume * Book Price
· · Total Sales = Book Sales + Record Sales
· · · Profit = Total Sales - Total Expenses
Total Sales
Total Expenses
Profit
Profit = Total Sales - Total Expenses
bookreed.MDL
CALC
READY

Figure 6.8: The Formulas branches view

As you can see, some choices are omitted from the Roots and Branches star menus because they do not make sense. All allow you to look more closely at the variables that make up your model and how they interconnect.

The Table View

■ Unlike the Diagram and Formulas views, the Table view shows only one variable at a time. The Table view shows you a table of the values of one variable for many different periods. The table is arranged in columns, with the date of the period and the value of the variable for that period. If the variable is a monthly variable, you see a list of months with the value of the variable for each month. If the variable is quarterly, yearly, or has some other periodicity, you see a list of the quarters, years, or another period (see Chapter 9, "Javelin and Time," for how to control the period of a variable). At the top of the view, Javelin also displays the formula for the variable.

To see the Table view of Total Sales, give the command / **V**iew **T**able. Javelin asks the name of the variable you want to see. Type the name or use the Spell key, and select Total Sales. A list of months and values appears (see Figure 6.10). To see another variable, use the Goto key or give the / **V**iew **T**able command again and type in a different variable name.

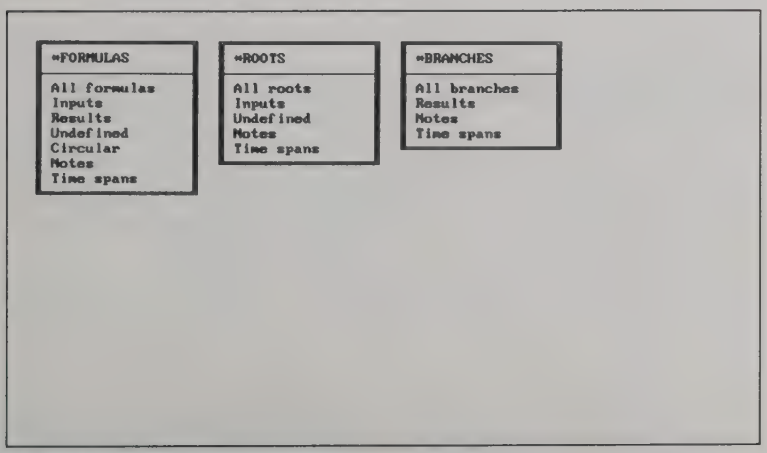


Figure 6.9: The Formulas star menus

You can enter or change data in the Table view if the variable you are looking at is an input variable. If it is a calculated variable, you cannot change the values—Javelin will tell you *Cannot add or change data; variable is already defined by formula.* For the variable Total Sales, Javelin will not let you enter values, because Total Sales is calculated from Book Sales and Record Sales.

Bring up a Table view of the variable Book Volume and change the data: press F5 and type

Book Volume

Javelin displays the view. To change a value, you can move the cursor to the month you want to change and type a new value.

To try this, move to Jan 1987, then type

50

As you type each number and press ⟵, the cursor moves to the next month (or period) automatically. To repeat the number you just entered, press ⟵ again. If you find you have skipped a number as you are typing, press the Ins key. This pushes all the following numbers down by one period. The period you are on and the next period both contain the number you originally typed in for the current period.

The easiest way to understand this is to try it. Your cursor should be on Feb 1987. Look carefully at the value for Feb 1987 and Mar 1987. Look also at the end of the table, Dec 1987. Insert a value between January and February by pressing Ins.

The value for Feb 1987 should be copied to Mar 1987, the old value for Mar 1987 pushed to April, and so on all the way through the value for December, which has been pushed to Jan 1988. Javelin won't insert a blank value for February: it copies the January value. In fact, Javelin never lets you have a blank value between two other values. We can illustrate this another way.

If you enter a value for March 1988, what will happen to February 1988? Move to Mar 1988. Type

1000

In Javelin PLUS, your screen should look like Figure 6.11. In Javelin 1.1, the value #NA appears for February (as in many spreadsheet programs, NA stands for *Not Available*). Any formula using an NA value will also be NA.

(Javelin gives you the alternative of setting missing values to zero, as well as a way to turn all the NAs in a series into zeros. See Chapter 8, "Formulas and Functions.")

Now you have entered several erroneous values in Book Volume: Feb 1988, Mar 1988, and Mar 1987. The Del key removes a number from the series and moves all the following numbers back up one time period.

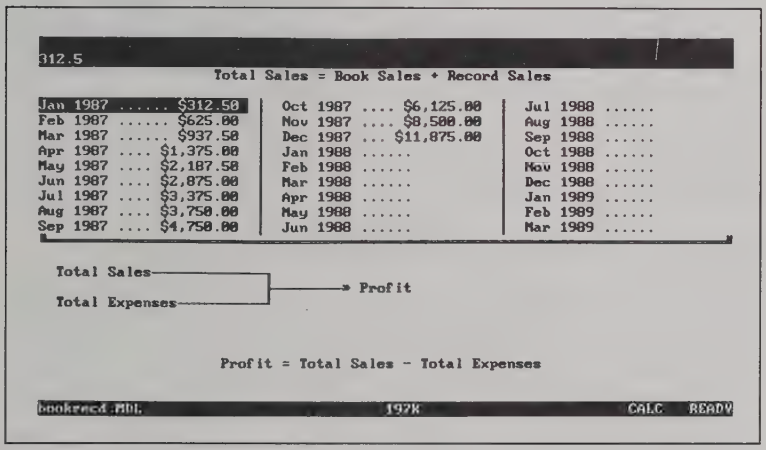


Figure 6.10: Table view of Total Sales

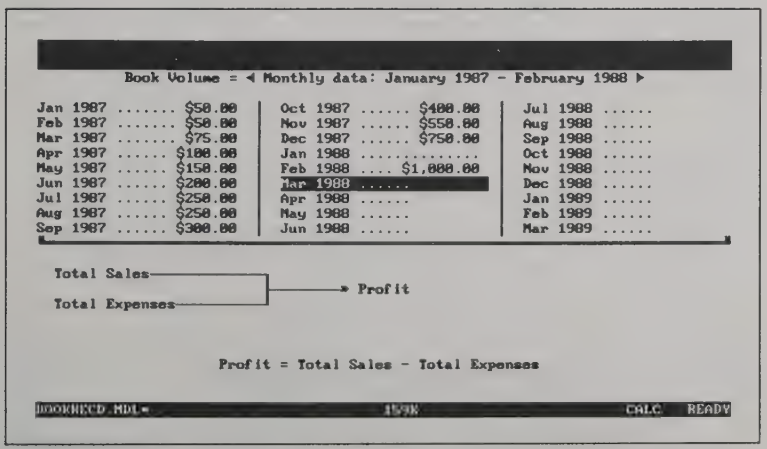


Figure 6.11: Book Volume with a blank value

Move to Feb 1988, then press Del. Notice that the value 1000 moves up to Feb 1988 and March 1988 is blank. (It is blank rather than #NA because it was the last number in the series. By deleting February, you have made the series shorter by one time period.)

To clean up the rest of these values, press Del, move to Mar 1987, press Del again, and finally press Home. Again, when you delete March 1987, the value for April moves to March, the value for May moves to April, and so on down to the end of the series: January 1988 moves to December 1987.

The Table view is the simplest way to see the story (its formulas and values) on one variable. See Table 6.3 for a summary of how to edit data in the Table view.

The Chart View and the Quick-graph View

■ These views are very similar; both are graphic representations of the Table view. Like the Table view, they show the values for one variable, but instead of showing them as numbers, the Chart view displays a bar chart and the Quick-graph view displays a line graph in which the height of the bar or

Key name	Function
Del	Deletes the value the cursor is on and moves the rest of the values up.
Ins	Repeats the value where the cursor is, moving all the rest of the values down one. (If the cursor is on the first value for the variable, it just moves the rest of the values down one.)
Arrow keys	Move the cursor from value to value.
Home	Moves the cursor to the first value in the table.
End	Moves the cursor to the last value in the table.
F2 (Edit)	Edits the value the cursor is on.
F5 (Goto)	Displays the values for a different variable.

Table 6.3: How keys work in the Table view

line corresponds to the values for the variable. As with the Table view, you can enter and edit the values if the variable is an input variable.

Although the Chart view displays a bar chart, Javelin constructs it using the PC's extended graphics characters, rather than using the graphics mode. This means that even if your screen cannot display graphs (for example, if you have an IBM Monochrome Adapter and Display), you can still see the Chart view. This also means that you can have the Chart view in one window and another view in the other window.

Because there is no way to draw sloping lines on the PC screen with characters, Quick-graph uses the PC's *graphics mode* to display its graph. This means that if you have only an IBM Monochrome Adapter, *you cannot use Quick-graph;* the IBM Monochrome Adapter cannot display graphics. Because it is difficult to mix the text already on the screen with new graphics, the Quick-graph view also takes over the whole screen: you cannot have a Quick-graph view in one window and another view in the other.

Let's look at the Chart view for Book Volume and compare it to the Table view. You should still have the Table view for Book Volume in the bottom window. (If you only have one window, use the command / **W**indow **2**-windows to display two. If neither window shows the Table view, give the command / **V**iew **T**able Book Volume.)

Move the cursor to the other window to leave the Table view on the screen. Press F6 and, to display the Chart view of Book Volume, give the command / **V**iew **C**hart. Javelin prompts for the name of the variable to chart. Now press ⇆ and select Book Volume.

Figure 6.12 shows the Chart view—it is a bar chart of the variable. The horizontal axis shows time, in months or whatever period the variable is set for (if the period is months, just the first letters of the month names are shown). The vertical axis shows the values for the variable. At the top of the window, Javelin shows the formula for the variable.

For those with graphics monitors: look at the Quick-graph view of the same information. There are two ways of showing a Quick-graph. One way is to give the command / **V**iew **Q**uick-graph and tell Javelin which variable you wish to see (as usual, you can press ↵ and Javelin will use the current variable). Instead, press F8, the Quick-graph key, and Javelin will go ahead and show you a Quick-graph of the current variable.

You see the same information as for the Chart view, except that it takes up the whole screen, and you have a line graph instead of a bar chart. Figure 6.13 shows the Quick-graph view of Book Volume. To return to the normal

Javelin screen, press Esc. You can change back and forth between the Javelin windows (now showing the Chart and Table views) and the Quick-graph view by hitting the Quick-graph key and the Esc key.

How did Javelin decide what scale to use on the left sides of the chart and graph? In each case Javelin chose the left scale depending on the variable's values. There is no way to control the scale in Chart and Quick-graph.

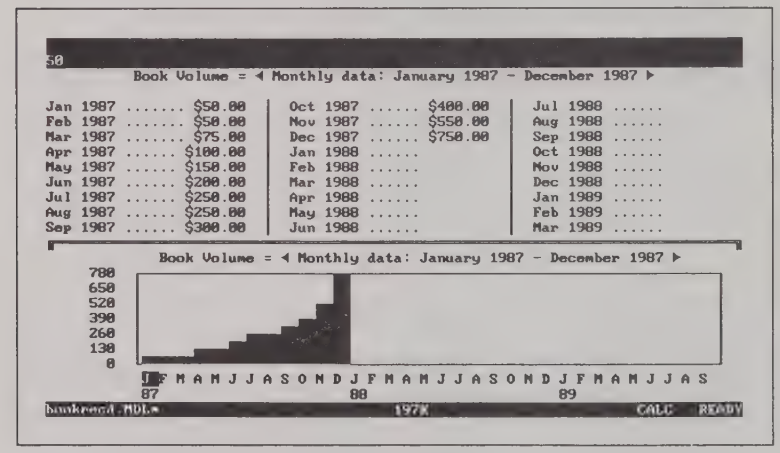


Figure 6.12: Chart view of Book Volume

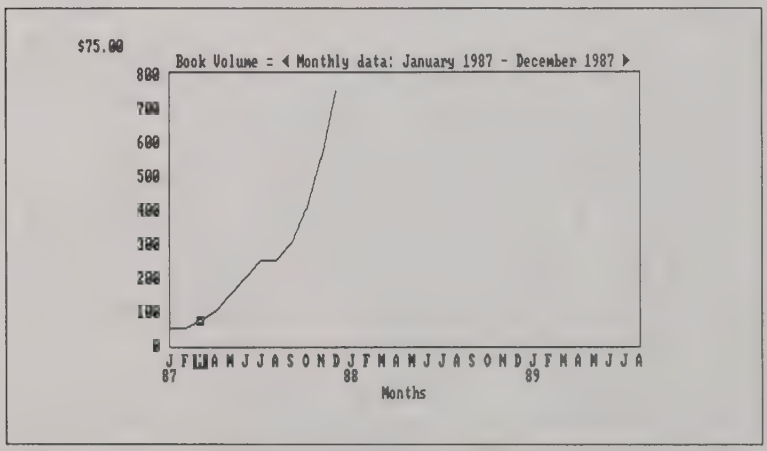


Figure 6.13: Quick-graph view of Book Volume

(In the Graph view, you can control many aspects of your charts and graphs. See Chapter 12, "Graphing.")

■ *Entering and Editing Data*

If the chart or graph represents an input variable, you can not only look at it but also enter and edit data. The cursor starts out on the first data value—note that the first label on the horizontal axis is highlighted. You can move the cursor from bar to bar or point to point with the → and ← keys.

When the cursor is on a value (assuming that the variable is an input variable), you can change the value by using the ↑ and ↓ keys. Notice as you do this that the number displayed in the upper-left corner of the screen changes to reflect the new value. The data increase or decrease by an amount determined by Javelin, depending on the left scale.

However, *be careful when you change values this way.* As long as you do not approach the top or bottom of the graph or chart, you can return the data to its original value just by using the ↑ and ↓ keys and watching the value in the upper left. But when you get to the top or bottom, Javelin has to redraw the left scale. When it does this, it changes the amount by which ↑ and ↓ affect the value. This means that if you hit ↑ four times, and after the fourth stroke Javelin redrew the left scale, if you hit ↓ four times, *you will not return to the original value.*

Fortunately, if you remember the original value, you can type it and press ⏎. Javelin will replace the data with the value you typed. The PgUp and PgDn keys also work, and they increase or decrease the bars by twice as much. The same cautions apply as for the ↑ and ↓ keys. You can also insert and delete values using the Ins and Del keys just as in the Table view (see Table 6.4 for how the keys work in these views).

For a long series of data, your Chart or Quick-graph view may not fit across the screen. In this case, Javelin shows as much as it can, and as you move the cursor toward the right edge of the screen more of the chart comes into view.

Make sure the Chart view is on the screen (hit Esc if you are in the Quick-graph view). Move to Sept 87. Take a look at the September 1987 entry for the Table view in the other window. Now press ↑ twice. Notice that the value for September 1987 changes when you change the height of the bar in the Chart view. To return September 1987 to its original value, press ↓ twice.

The Chart and Quick-graph views are the easiest way to work with a variable for which you have a graphic idea of how the data will look.

The Notes View

■ One objective of Javelin is to help you make models that are clear and easy to understand. This is the idea behind organizing the model into variables with formulas or data, and using graphs, worksheets, and tables as views of the underlying model. But no matter how well you design your

Key name	Function
Del	Deletes the value the cursor is on and moves the rest of the values to the left.
Ins	Repeats the value where the cursor is, moving all the rest of the values one place to the right. (If the cursor is on the first value for the variable, it just moves the rest of the values one place to the right.)
←	Moves the cursor to the previous value.
→	Moves the cursor to the next value.
Home	Moves the cursor to the first value in the series.
End	Moves the cursor to the last value in the series.
↑	Edits the value the cursor is on by making the bar or line higher and the value greater.
↓	Edits the value the cursor is on by making the bar or line lower and the value less.
PgUp	Same as ↑ except that it increases the bar or line by twice as much.
PgDn	Same as ↓ except that it decreases the bar or line by twice as much.
F5 (Goto)	(Chart view only.) Displays the values for a different variable.
Esc	(Quick-graph view only.) Exits from the Quick-graph view and returns to the view(s) previously on the screen.

Table 6.4: How keys work in the Chart and Quick-graph views

model, it always helps to be able to describe it in words. For instance, behind each variable is a story: where did the data come from; why is the formula written this way and not some other way; what are the limitations of the data; how accurate do you think they are? These are the kinds of information that appear in the footnotes to many statistical tables. In Javelin, these footnotes appear in the Notes view; each variable (whether data or formula) may have a note associated with it. The Notes view allows you to enter, edit, and read the notes about the variables.

Enter a note about the Record Volume variable, using the command **/ View Notes**. Press ⟶| and select Record Volume. You should see the heading Notes for Record Volume: and a blank space for the notes (see Figure 6.14). A small blinking cursor, the cue, marks your place on the screen.

What you have here is a very simple text editor, less powerful than the simplest word processor but sufficient for notes about a variable or your model. For example, type in this note:

> **These volumes are based on Fred's estimates. He included seasonal adjustments for increased Christmas sales.**

Your notes can be quite long, more than will fit on the screen. See Table 6.5 for a list of the editing keys available when you are in the Notes view.

It is a good idea to make notes of the sources of your data, of the assumptions you are making as you set up the model, and of how you

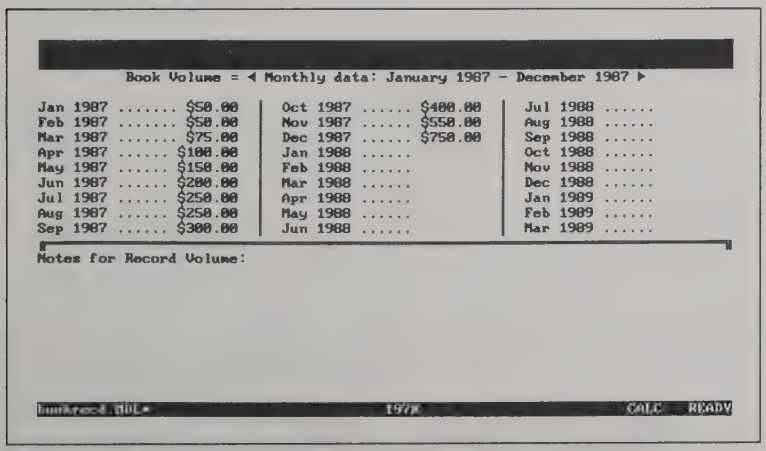


Figure 6.14: The Notes view

arrived at your formulas. This will make it easier to review the model later and to interpret the data correctly. You can print a note by using the command / **P**rint **C**urrent view; your note appears on the page with the heading Notes for variable name.

Once you have created a note, you can edit it by viewing it again and making changes. To get rid of a note, use the command / **C**lear **N**ote. You will be asked which variable you wish to clear the note from; pressing ⟵ clears the note from the variable you are looking at.

Key name	**Function**
Arrow keys	Move the cue one space left or right, or one line up or down.
Del	Deletes the character the cue is under.
⟵	Deletes the character immediately to the left of the cue.
Home	Moves the cue to the beginning of the line.
End	Moves the cue to the end of the line.
PgUp	Moves the cue to the previous page of the note (for long notes).
PgDn	Moves the cue to the next page of the note (for long notes).
⟵	Moves the cue to the beginning of the next line.
Ctrl-Home	Moves the cue to the beginning of the note.
Ctrl-End	Moves the cue to the end of the note.
Ctrl-K	"Kills" characters after the cue on the same line and stores them in the *kill buffer.*
Ctrl-Y	"Yanks" characters from the kill buffer and places them where the cue is. Ctrl-K and Ctrl-Y can be used together to move text from one part of a note to elsewhere in the same note, or from one note to another.
Ctrl-Q	Used for typing characters that have a special meaning to Javelin, for example, the / key. If you are typing a note and hit the / key, Javelin brings up the command menu. To put a / into your note, press Ctrl-Q followed by the special key, in this case /. The / will appear as a character in the note rather than making the command menu appear.

Table 6.5: The editing keys in the Notes view

The Other Views

There are four other views through which you can look at your model: Worksheet, Graph, Errors, and Macro. The Worksheet view and the Graph view are both powerful enough to merit their own discussions, in Chapters 7, "Defining Variables and Entering Data Using Worksheets," and 12, "Graphing." The Errors view and the Macro view are each useful in connection with a specific action: diagnosing errors in your model and creating repetitive procedures (macros). They are described in Chapters 11, "Special Values and Diagnosing Problems," and 14, "Macros." Here is a quick look at each view.

The Worksheet View

The Worksheet view is the view that looks the most like a traditional spreadsheet program. Javelin worksheets can be used both to build your models and to prepare reports. In Chapter 7, you will see how to use the Worksheet view to create and modify variables. In Chapter 13, "Advanced Worksheets and Printing," you will see how to format worksheets exactly as you want them and how to set up sophisticated worksheets.

The Graph View

The Graph view allows you to view graphs that have been defined using the command / **D**efine **G**raph. You can create bar graphs, stacked bars, line graphs, pie charts, and other graphs. The Graph view differs from the Chart view and Quick-graph view in several ways:

- You can view several variables in the same Graph view. The Chart and Quick-graph views display only one variable at a time.
- You cannot enter or edit data in the Graph view.
- You can set up titles and legends, choose symbols and colors, and control scaling and other aspects of the Graph view to make a graph more readable.

Creating and viewing graphs is covered in detail in Chapter 12.

■ The Errors View

When you create a complex model, the Errors view gives you the tools you need to trace and fix problems in the model, such as undefined variables or incorrect use of time periods. For details about the types of errors you may run into and how to solve them, see Chapter 11.

■ The Macro View

Javelin includes a facility to record and play back the keystrokes you press to do repetitive tasks. These keystroke recordings are called macros, and the Macro view is used to create and edit them. For more information, see Chapter 14.

The Sync Key

■ If you are using two windows to look at different views of your model, you can tell Javelin to keep the two windows "in sync," that is, when you move to a variable in one window, Javelin should display the view of the same variable in the other window.

The Sync key is F3. When you press it, the SYNC indicator appears on the status line, showing that Javelin is keeping the windows in sync. The most common use for the Sync key is to allow you to move from variable to variable in the Diagram view, while the Table view in the other window shows the variable you are on. Try it with your BOOKRECD model.

Move to the bottom window and display a Diagram view of Book Expenses. (If you are not in the bottom window, press F6 first.) Give the command / **V**iew **D**iagram Book Expenses. You should still have a Table view of the same variable in the top window. If you don't, switch windows, display a Table view, and switch back, by pressing F6, giving the command / **V**iew **T**able Book Expenses, and pressing F6 again.

Now press F3 to tell Javelin to keep the two windows in sync. The SYNC indicator has appeared next to the READY indicator on the bottom line of the screen. Move around in the Diagram view and watch what happens to the Table view in the other window. Press → twice, then ← once. The Table view changes to display the same variable that your cursor is on in the Diagram view.

To "unsync" the windows, press F3 again. SYNC also turns off if you switch windows using F6 or use the command / **W**indows **S**witch.

Sticky Sync was added in Javelin PLUS. Sticky Sync also turns off when you switch to the other window. But, unlike regular Sync, when you switch *back* to the original window Sync comes back on. This is useful if you want to scan through the variables in your model using one view, but work on those variables in another. You turn Sticky Sync on and off by holding the Alt key and pressing F3.

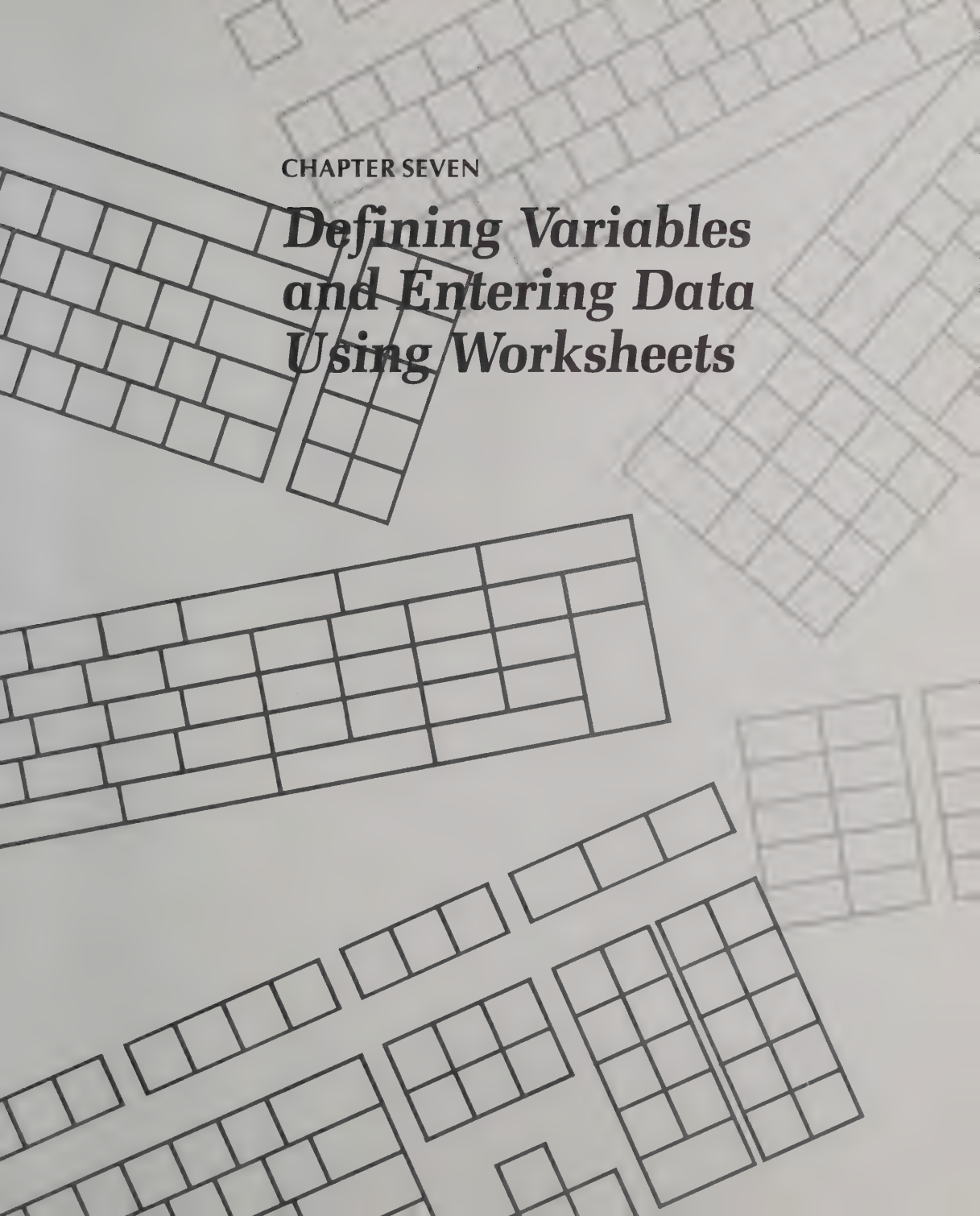
Conclusion

■ Javelin offers a number of ways to look at a model. Just as its name implies, the view itself is not your model, but a different way of looking at (and changing) your underlying model. Table 6.6 summarizes what you can do in each view.

View	Formulas	Data	Notes	Macros
Diagram	Yes	No	No	No
Formulas	Yes	No	No	No
Table	Yes	Yes	No	No
Chart	Yes	Yes	No	No
Quick-graph	No	Yes	No	No
Notes	No	No	Yes	No
Worksheet	Yes	Yes	No	No
Errors	Yes	No	No	No
Macro	No	No	No	Yes
Graph	No	No	No	No

Table 6.6: What to enter or edit in each view

Next, we'll look at Javelin's most powerful view—the Worksheet view.



CHAPTER SEVEN

Defining Variables and Entering Data Using Worksheets

The Worksheet view is quite different from other Javelin views. It is the only one that gives you the ability to display data from several variables as well as move, format, and copy data.

Uses of the Worksheet

- There are several tasks for which the worksheet is particularly useful.
  - Reporting: Use the worksheet to set up and print tabular reports. No other view allows you to lay out columnar reports that include the values of several variables.
  - Analysis: Use the worksheet to analyze your model by displaying data from several variables side by side, or by converting variable values to other time periods (for example, displaying quarterly totals for a monthly variable).
  - Data entry: Use the worksheet to enter data for several variables at once. The Table view displays only one variable at a time.
  - Variable creation: When you are creating calculated variables with a regular structure, use the worksheet to copy the repetitive formulas without repetitive typing. For example, if

    Lemon Income = Lemon Unit Sales + Lemon Price

    and

    Lime Income = Lime Unit Sales + Lime Price

    you can type in the Lemon formula and copy it to create the Lime formula.

To explore the uses of the worksheet, let's recreate the BOOKRECD model, doing all the data and formula entry in the Worksheet view. Then let's create some additional reports that show how to format the numbers on your reports and to use the worksheet to do date conversions.

■ Note to 1-2-3 Users

The Worksheet view works very differently from a 1-2-3 spreadsheet, but many of the keys work the same, including the cursor motion keys, the Edit key (F2), and the Calc key (F9). In addition, several commands in the Worksheet view work like their 1-2-3 counterparts, including the commands for copying information from one part of the worksheet to another. Table 7.1 shows the corresponding keys and commands.

Parts of the Worksheet

■ Although a Javelin worksheet looks like a spreadsheet, it is actually quite different. A Javelin model can contain many different worksheets, each of them identified by a name displayed at the top of the Worksheet view.

A worksheet is made up of a grid of vertical *columns* and horizontal *rows*. The intersection of a row and a column is called a *cell*. The cell in the top row and the leftmost column is called the *home cell*.

Each worksheet has *borders* and a *body*. The borders are the row(s) at the top and the column at the left of the worksheet where you tell Javelin what dates and variables you want to use; the body is the rest of the area in the worksheet where your data appear (see Figure 7.1).

There are actually three border areas: the *left border* (leftmost column), the *top border* (top row or rows), and the *corner border* (upper-left corner). Javelin decides what to display in each cell by combining the information in the top border above the cell, in the left border to the left of the cell, and in the corner border above and to the left of the cell. (You can override this for individual cells, as you'll see later.) This means that each top border cell affects its entire column, each left border cell affects its entire row, and the corner border potentially affects the whole worksheet.

Like Javelin's other views, the Worksheet view can be larger than what you see on the screen. The size of a worksheet is limited only by the amount of free memory in your computer.

Using borders lets you set up worksheets quickly and easily, because you have only to fill in the borders; Javelin automatically fills in the body cells for you.

KEYS:

Javelin	1-2-3	Comments
Arrow keys	Arrow keys	Same.
F1	F1	Same (Help).
F2	F2	Same (Edit).
F5	F5	Same (Goto), using variable names and dates instead of cell addresses and range names.
F9	F9	Same (Calc).
PgUp	PgUp	Same.
PgDn	PgDn	Same.
Home	Home	Moves to upper-left corner in border cell.
Ins	/ Worksheet Insert	In left border, inserts a row. In top border, inserts a column.
Del	/ Worksheet Delete	In left border, deletes a row. In top border, deletes a column.
/	/	Same (display command menu).
Esc	Esc	Same (back up one level when giving a command).

COMMANDS:

Javelin	1-2-3	Comments
/ *Worksheet **C**opy	/ Copy	Relative copy when copying body cells.
/ *Worksheet **D**ata copy	/ Range Value	Copies data only, not formulas.
/ *Worksheet **W**idth	/ Worksheet Column-Width	Changes the column width of one column.

Table 7.1: Lotus 1-2-3 keys and commands, and their Javelin counterparts

SETTINGS ON SETTINGS SHEETS (S.S.):

Javelin Settings Sheet and Setting	1-2-3 Command	Comments
Worksheet S.S.: Column width setting	/ Worksheet Global	Same (sets width for all columns).
Model S.S.: Format	/ Worksheet Global Format	Same (sets default numeric format).
Variable S.S.: Worksheet S.S.	/ Range Format	Similar.
Format S.S.; Format setting		

Table 7.1: Lotus 1-2-3 keys and commands, and their Javelin counterparts (cont)

■ *Displaying the Worksheet*

When you start up Javelin PLUS, you see a Formulas view in the top window and a Worksheet view in the bottom window.

When you start up Javelin 1.1, you see a Diagram view in the top window and a Table view in the bottom window. Move to the top window by pressing F6, and give the command / **V**iew **F**ormulas **A**ll, so you'll see the list of formulas grow as you create the model. Press F6 to return to the bottom window and give the command / **V**iew **W**orksheet. Javelin prompts you to give the worksheet a name. If you don't give it one, the worksheet will be called WORKSHEET, (Worksheet 1 in Javelin PLUS). Call this one Sales, because you will start out by defining the sales part of the model. Type

Sales

This brings up a blank worksheet, which looks like Figure 7.1. The top line of the worksheet shows the name of the worksheet, centered. The top border is directly below the worksheet's name. The borders are highlighted and the body is in normal text. The cursor is an underline on non-graphics screens and a colored box on color screens.

■ Moving Around the Worksheet

You can use the cursor keys and others to move around the worksheet. When the worksheet first appears, the cursor is in the cell in the upper-left corner, the home cell. See Table 7.2 for what the keys do in the Worksheet view.

Defining Input Variables in the Worksheet

■ To create the variables that make up your model, you must enter information into the borders of the worksheet to indicate the names of the variables with which you are working and the time periods (in this model, months) for which you want to enter or calculate values. Then you can enter the data for the input variables and the formulas for the calculated variables.

Javelin doesn't care which border contains the dates and which contains the variable names, but for this example it is more convenient to enter the dates across the top and variable names down the side.

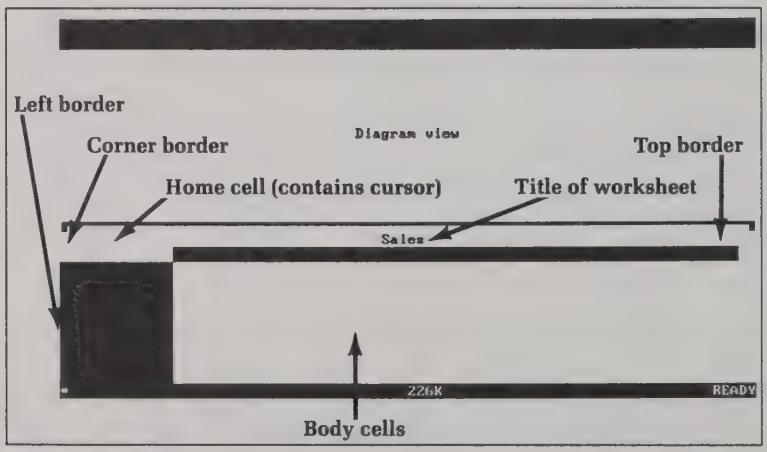


Figure 7.1: Parts of a worksheet

■ *Entering Dates in a Border*

Move the cursor to the first cell in the top row (not the corner cell—the one immediately to its right) by pressing Home and then →. Enter the first date, in this case January 1987, by typing

jan87

You can spell out January 1987 if you want, but Javelin is prepared to handle the shorter form, too (see Figure 7.2).

Since you frequently have a series of dates to enter, Javelin provides a way to make date entry easier. If you press ↵ in a blank border cell and there is a date just before it (to its left if you are in the top border, or above it if you are in the left border), Javelin automatically fills in the next date in the series.

←	Move left one column.
→	Move right one column.
Home	Return to the home cell.
↑	Move up one row.
↓	Move down one row.
PgUp	Move up one screenful of rows.
PgDn	Move down one screenful of rows.
End (followed by any arrow key)	Move to the end of the worksheet in the direction of the arrow key.
Ins	When in left border, insert a blank row. When in top border, insert a blank column.
Del	When in left border, delete the current row. When in top border, delete the current column.
Ctrl-→	Move right one screenful of columns.
Ctrl-←	Move left one screenful of columns.

Table 7.2: What keys mean in the Worksheet view

Move the cursor to the next cell to the right (using →) and press ↵ again. Javelin automatically fills in the next date, Feb 1987, and moves the cursor to the next cell.

If you press ↵ several more times, Javelin fills in dates across the top border. Since our model runs for a year, keep pressing ↵ until it fills in the date for December, as in Figure 7.3. As you move to the right, Javelin scrolls the worksheet so that you can easily move around a worksheet that is larger than the screen.

■ *Defining Variables in the Worksheet*

Now enter the variable names in the left border for the two time-series input variables in this model: Book Volume and Record Volume. As you enter each variable name in the left border, Javelin creates that variable in the model. You can then enter the values for each month by typing the numbers into the body of the worksheet under the border entry for each month. Move to the first cell in the left border by pressing Home and then ↓.

Start by entering the values for Book Volume. In the border, type

Book Volume

Then press →, moving into the body of the worksheet. Notice that on the information line, Javelin displays the variable name Book Volume:, which means that Javelin has created the variable and is ready to do something with it.

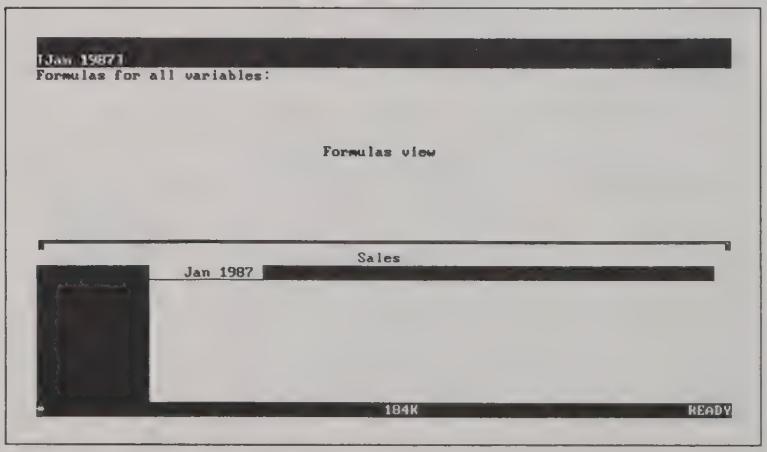


Figure 7.2: Entering a date in the worksheet

■ *Entering Values in the Body of the Worksheet*

Type in the January Book Volume value,

25

When you press ⟵, several things happen. First, the value appears in the cell. Second, on the information line, Javelin now displays

Book Volume @ January 1987: 25

The symbol @ (called the "at" sign) signifies the value of a variable at a particular date or period. It tells you that Book Volume has been defined as a monthly variable and that this is its value for January 1987. Finally, the top half of the screen, which displays a Formulas view, shows your new variable and tells you that it is defined as monthly data with values from January 87 to January 87. The screen should now look like Figure 7.4.

Move the cursor to the right and enter the value for February. Type

50

The Formulas view in the top window now confirms that the monthly data run through February.

You can enter the rest of the values for Book Volume this way. Before you do so, arrange to enter the values for Record Volume at the same time.

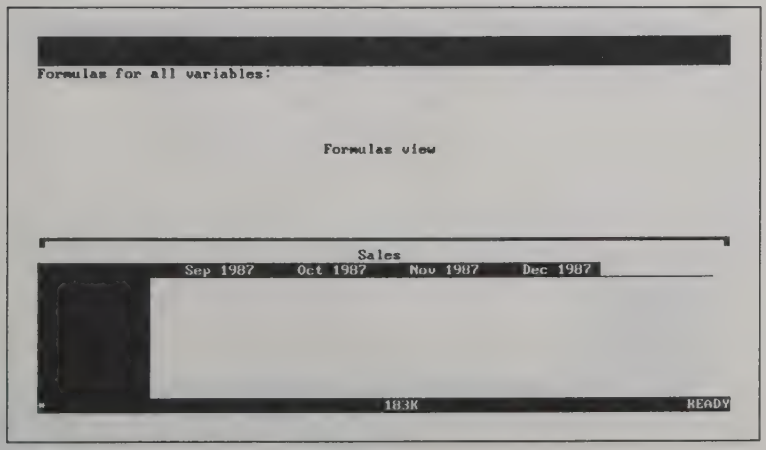


Figure 7.3: Worksheet after entering dates for a year

Move back to the left border (press ← twice) and down to the next row. Enter the name of the next variable,

Record Volume

This name doesn't quite fit in the border, but even though it looks peculiar Javelin understands what you are doing. You can widen the border later so this looks better.

Enter the January and February values, in this case 10 and 20. Use → to move into the column for each month before typing the value. The information line confirms that you have entered monthly values for Record Volume and that its value at February 87 is 20. The Formulas view in the top window now shows both Book volume and Record volume.

To enter the data for the rest of the year, type in alternate values for book and record volumes. Move the cursor to the cell for Book Volume at March, enter the March values, and then move down to enter the Record Volume value for March. Javelin saves you a little typing by not requiring ↵ between values if you use the cursor keys to move from cell to cell. To enter the March values, press ↑ and →, type

75

and press ↓, then type

30

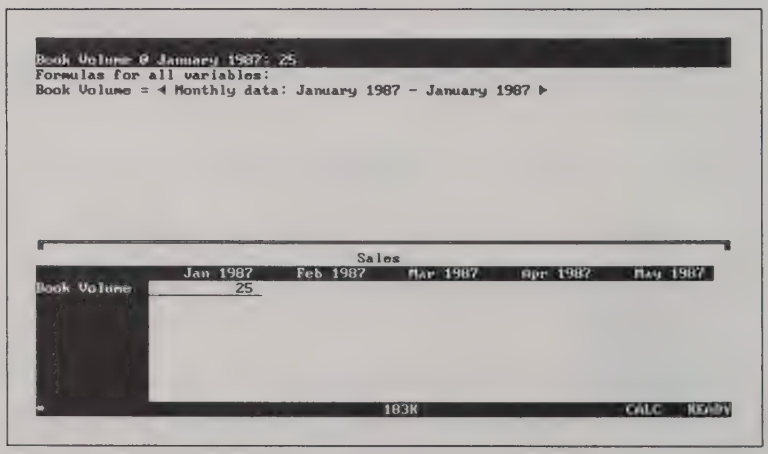


Figure 7.4: Worksheet after entering a value

As soon as you press ↓, it enters the value 75 and moves the cursor down to the next row. You can now quickly enter the values for April and May (refer to Table 7.3 for the numbers to enter).

The rest of the worksheet is off the screen. Move the cursor to the May column and press →. Notice that the top border and the body of the worksheet scroll left, so you see the columns for February through June, but the border on the left doesn't change. As you scroll around the worksheet, Javelin always displays the borders that affect the cell where the cursor is, so you can tell where the values displayed in the worksheet came from. This resembles the *locked titles* feature familiar to spreadsheet users, but since Javelin distinguishes between border cells and body cells, the borders stay visible automatically.

Finish entering the values for book and record volume through December using the values in Table 7.3. When you are done, your screen should resemble Figure 7.5.

Inserting and Deleting Parts of a Worksheet

■ At this point, we will show you the single clearest demonstration of how the Javelin worksheet differs from a spreadsheet—what happens if you delete part of the worksheet.

■ Deleting Rows and Columns

The Del key is used to delete a row or column from your worksheet. If your cursor is in the top border, you delete the column you are in. If your cursor is in the left border, you delete the row.

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Book Volume	25	50	75	100	150	200	250	250	300	400	550	750
Record Volume	10	20	30	50	85	110	120	150	200	250	350	500

Table 7.3: Values for Book Volume and Record Volume

Let's delete the Record Volume row. Move the cursor back to the left border cell that contains Record Volume and delete the row. To do this, press Home, ↓ twice, and Del.

The label Record Volume disappears, as does the entire row of data to its right. If there were rows below it, they would move up to fill the gap. If this were a spreadsheet, you would just have discarded half the data that you typed in, and you would be justifiably mad at this book for wasting your time in useless typing. Fortunately, since the worksheet is not a spreadsheet, you can remain calm.

Move the cursor down two rows and type into the border

Record Volume

Lo and behold, your entire row of record volume data appears again, two rows below where it used to be. Your typing was not in vain.

This underscores the most important difference between the worksheet and a spreadsheet. In a spreadsheet your data exist only in the cells where they appear. In Javelin, your data reside in the central information base for your model, and the worksheet is a window into it. All ten Javelin views display information from this central base. Generally speaking, the only way to delete data is through the command / **C**lear menu, and you can delete and

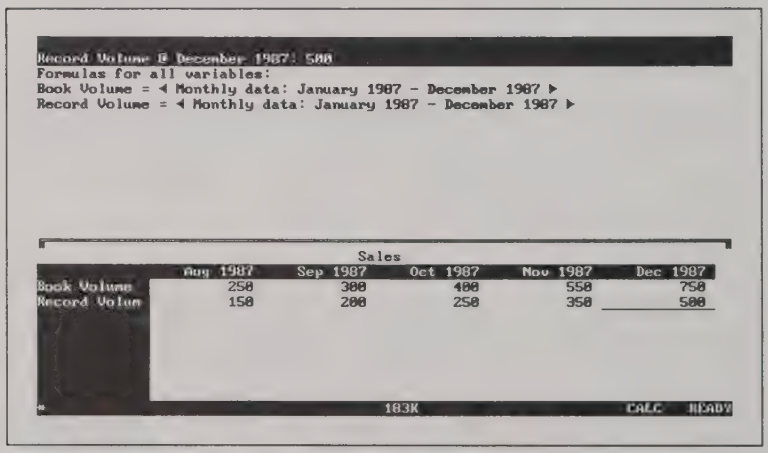

Record Volume @ December 1987: 500
Formulas for all variables:
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Sales
Aug 1987 Sep 1987 Oct 1987 Nov 1987 Dec 1987
Book Volume 250 300 400 550 750
Record Volu 150 200 250 350 500
183K
CALC
READY

Figure 7.5: Worksheet after entering data for a year

move things on a worksheet without fear of accidentally losing the underlying data.

You can do the same thing with the dates in the top border. Move the cursor to the cell in the top border that displays Feb 1987 and delete the column. Press Home, → twice, and Del.

The column disappears and the columns to the right move over to fill in. Your February data have not been deleted, since the data reside in the central information base.

■ *Inserting Rows and Columns*

To insert a new blank row or column in a worksheet, you can use the Ins key. If you are in the top border, Ins adds a new column where your cursor is. If you are in the left border, it inserts a new row.

Press Ins: Javelin opens up a blank column between January and March. Now press ⟵ and Javelin automatically fills in a date (since the border cell to the left has a date), and your February data are called up again from the central information base for this model.

Defining Constants in the Worksheet

■ So far, you have entered two time-series input variables, Book Volume and Record Volume. Next, define the two constants that go with them, Book Price and Record Price.

To define a constant, move your cursor into a border cell and type the formula. The name of the constant variable appears in the border cell, and the value of the variable will be displayed in each cell controlled by that border cell.

Move down into the left border and type the formulas for the first variable. Let's put Book Price just below Book Volume and Record Price below Record Volume. Type

Book Price = 7.50

The value for Book Price is displayed at every month. Since Book Price is a constant, it always has the same value. Press → to move into the body of the worksheet. The information line says

Book Price: 7.5

This indicates that the variable has been defined as a constant, because no date is mentioned. Your worksheet should look like Figure 7.6.

Correcting Mistakes in the Worksheet

■ Let's say you made a typing mistake when entering the definition for a variable, in this case Record Price. Press ← and ↓ three times to move to the left border cell. Then type

Record Prince = 12.50

The variable is stored as Record Prince rather than Record Price. You could correct the variable name in the border easily enough, using F2, the Edit key.

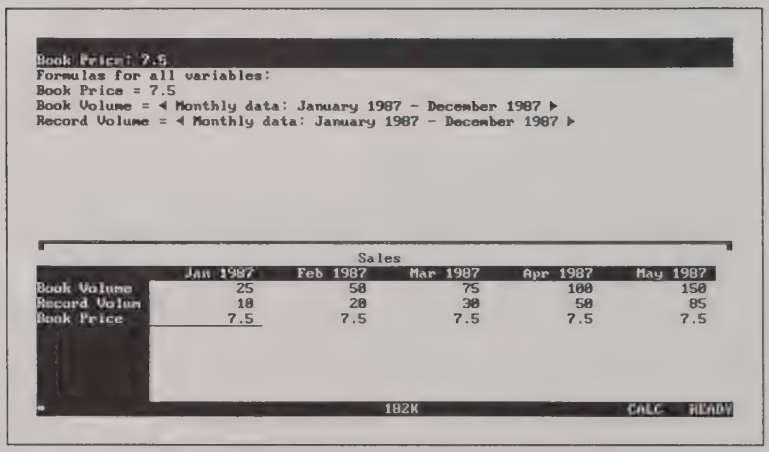

Book Price: 7.5
Formulas for all variables:
Book Price = 7.5
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Sales
Jan 1987 Feb 1987 Mar 1987 Apr 1987 May 1987
Book Volume 25 50 75 100 150
Record Volum 10 20 30 50 85
Book Price 7.5 7.5 7.5 7.5 7.5
182K
CALC READY

Figure 7.6: Book Price constant in the worksheet

However, this would define a new variable called Record Price, and Record Prince would still be defined.

■ *The / Clear Variable Command*

You should get rid of Record Prince to avoid future confusion. The command / **C**lear **V**ariable deletes variables. Be extremely careful when you use this command—it is one of the few ways to delete data from the central information base that makes up your model. If you delete a variable by mistake, there is no way to get it back into the model, unless you happen to have a version of the model saved on the disk from which you can retrieve it.

With all due caution, clear the variable you don't want. Your cursor should be on the left border of the Record Prince row. Give the command / **C**lear **V**ariable; Javelin prompts you to enter the name of the variable to delete and makes a guess.

```
Variable (Record Prince):
```

In this case the guess is right, so press ↵ to go ahead and clear it. The wrong variable is now gone for good. The label is still in the border, but no numbers show up in the body of the worksheet, which indicates that the variable is no longer defined.

■ *Changing a Variable Name in the Worksheet*

There is an easier way to fix this kind of typing mistake. Rather than deleting Record Prince and creating Record Price, you can change the name of the variable. As you saw in Chapter 6, most views have a star menu that has commands specific to the view. To use commands on a star menu, display the menu and select the option that begins with an asterisk, in this case *Worksheet. The command you want to use now is / *Worksheet **N**ame change.

Misspell the name of the variable again and change it to the correct name. Type

```
Record Prince = 12.50
```

Values reappear on the row. Make sure that your cursor is still on the row containing the variable whose name you want to change, Record Prince. To

use the Worksheet star menu, give the command / *Worksheet. Javelin now displays the Worksheet star menu, as in Figure 7.7. Table 7.4 shows a list of the / *Worksheet commands with descriptions of what they do. We will cover some in this chapter and the rest in Chapter 13, "Advanced Worksheets and Printing."

Select the command **N**ame change from the Worksheet star menu. Javelin asks for the correct name of the variable, and it displays the current (wrong) name.

Rename to: Record Prince

Move the cue to the right until it is under the *n* and press Del to remove the erroneous letter.

When you press ↵, Javelin puts the new name into the border and renames the variable to match. This changes the name of the variable in the central information base for the model.

The / *Worksheet **N**ame change command is considerably more powerful than we have seen here and can be used to rename whole groups of variables at once. We'll go into this in more detail in Chapter 13.

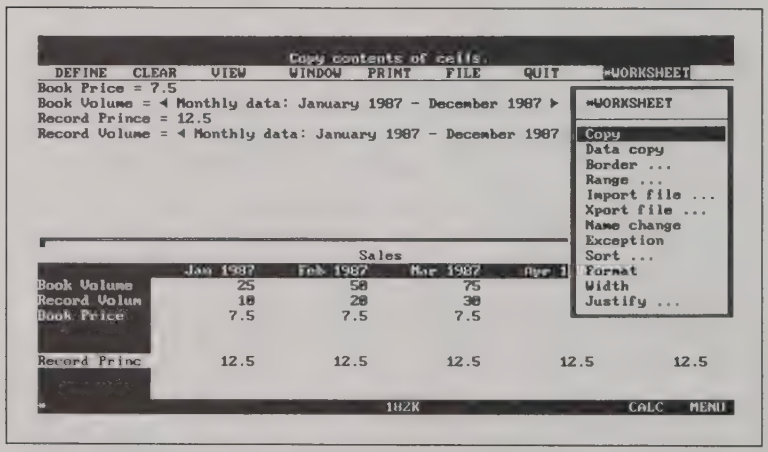


Figure 7.7: Worksheet star menu

Command	Description
Copy	Copies a range of cells from one part of the worksheet to another (see later in this chapter).
Data copy	Copies data only (not formulas) from one range of cells to another (see Chapter 13, "Advanced Worksheets and Printing").
Border Dates	Fills a border with dates (see later in this chapter).
Border Variable names	Fills a left border with variable names (see Chapter 13).
Border Insert border	Inserts a new set of borders (see Chapter 13).
Border Remove border	Removes a set of borders. You can't remove the top-left set of borders (see Chapter 13).
Border Expand border	Expands a top border by adding an additional row. This allows for superheadings on the top border, which work like indented headings in the left border (see Chapter 13).
Border Shrink border	Shrinks a top border by removing one row. You can't shrink a top border to less than one row (see Chapter 13).
Range . . .	Menu of commands that work with a range from the worksheet (see Chapter 13).
Import File . . .	Menu of commands that load a text file or Lotus 1-2-3 worksheet (see Chapter 19, "Using Javelin with Other Programs").
Name change	Changes the name of the variable the cursor is on (see later in this chapter).
Xport File . . .	Menu of commands that create a file readable by other programs (see Chapter 19).
Exception	Creates an exception cell in a body cell (see later in this chapter).
Sort Descending and Ascending	Sorts a range of rows into order (see Chapter 13).
Format	Displays the Format settings sheet for formatting rows, columns, or individual cells (see later in this chapter).

Table 7.4: Commands on the / *Worksheet menu

Command	Description
Width	Changes the width of a column (see Chapter 13).
Justify	(Javelin PLUS only.) Changes left, center, or right justification of strings. See Chapter 20, "Strings and Dates."

Table 7.4: Commands on the / *Worksheet menu (continued)

Defining Calculated Variables in the Worksheet

■ So far, you've created input variables, both time-series and constants. You can also use the worksheet to enter the formulas for calculated variables. To create a calculated variable, type the name of the variable into a border cell and type the formula into a body cell controlled by that border cell. Define Book Sales and Book Volume this way.

Define Book Sales first, using the blank row between Book Price and Record Volume. Press ↑ twice. Make sure you're in the left border and type

Book Sales

Then press → to move into the body of the worksheet. On the information line, Javelin displays Book Sales to show that this cell would contain a value for Book Sales but that there is not yet a formula or input data to define it. Now type the formula for book sales. In Javelin PLUS, you *must* precede the formula with an equal sign:

= Book Volume * Book Price

(In Javelin 1.1, do not type the = sign.)

At this point, #CALC appears across the row to show that Book Sales is defined by a formula, but that its values have not yet been computed. On the information line, Javelin displays

Book Sales: Book Volume * Book Price

to confirm that you have defined this variable by a formula. In the Formulas view in the top window, the formula for Book Sales appears.

Your screen should look like Figure 7.8. Calculate the worksheet to see these values by pressing F9. The answers appear.

Using Indented Borders

■ You could define Record Sales the same way, but instead try using indented borders to take advantage of similarities in the definition of book and record information. This can save typing and better organize the worksheet.

As you saw in Chapter 3, if you indent the words in the left border of a worksheet, Javelin combines the contents of the indented border cell with those of the closest cell above it that is less indented. This lets you group related variables together without repeating the words that the variable names have in common. Let's see how this works.

Clear away the variables created so far—remember that you can delete them from the worksheet without deleting them from the model itself. To erase the variable names from the left border, press Home, ↓, and Del six times.

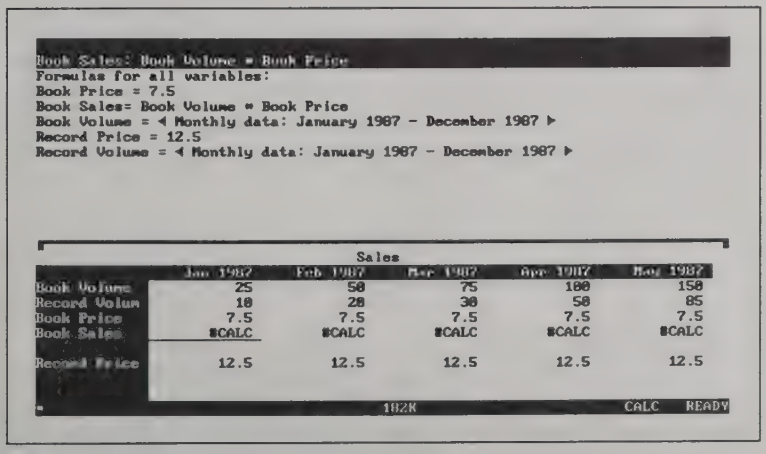

Book Sales: Book Volume * Book Price
Formulas for all variables:
Book Price = 7.5
Book Sales= Book Volume * Book Price
Book Volume = ◀ Monthly data: January 1987 - December 1987 ▶
Record Price = 12.5
Record Volume = ◀ Monthly data: January 1987 - December 1987 ▶
Sales
Jan 1987 Feb 1987 Mar 1987 Apr 1987 May 1987
Book Volume 25 50 75 100 150
Record Volum 10 20 30 50 85
Book Price 7.5 7.5 7.5 7.5 7.5
Book Sales $CALC $CALC $CALC $CALC $CALC
Record Price 12.5 12.5 12.5 12.5 12.5
102K
CALC READY

Figure 7.8: Worksheet with calculated variable

Now you can enter the variable names in the left border using indentation to show the organization of the variables in the model. First enter the book-related variables in the model, then the record-related variables.

In the left border, type

Book

and move down to the next row. So far Javelin displays no data because you have no variable named Book, just Book Volume and so forth. On the next line, indent two spaces, enter

Volume

and press ↓. The data for Book Volume appear.

Javelin sees that Volume is indented two spaces and scans up the border to find the closest border cell that is indented fewer spaces (or not indented at all). It finds the cell containing Book and puts the two parts of the name together to make the variable name Book Volume. Repeat the two spaces, type

Sales

and press ↓. The values for Book Sales appear, because Javelin combines the words Book and Sales to get the variable name. It does not combine Sales and Volume because they are indented the same number of spaces—Javelin looks above Sales for other names that are indented less and combines them. You could have indented both names just one space, and the results would have been the same. But if you had indented Volume one space and Sales two spaces, Javelin would have combined all three words and looked for a variable called Book Volume Sales. You can use multiple levels of indentation for more complicated reporting, as you will see in Chapter 13, "Advanced Worksheets and Printing."

Finally, create the last of the book variables, Book Expenses. After two spaces, type

Expenses

and press ↵ to enter the variable name (indented) into the border. No values appear in the body of the worksheet because Book Expenses is not

yet defined by input data or a formula. Enter the formula for Book Expenses in the body of the worksheet. Press → and type

= Book Volume * (0.75 * Book Price)

(Again, in Javelin 1.1, do not type the = sign.) Javelin enters the formula. The Formulas view in the top window is updated to show the new formula. Push the Calc key, F9, to calculate the values for Book Expenses. The screen now looks like Figure 7.9.

Now that you have the three Book variables, continue with the Record variables. Move back to the left border and down two cells, to leave a blank line between the book and record sections of the worksheet. Javelin doesn't require the blank line, but it makes the worksheet look better. Now enter the name into the border by typing

Record

*Creating New Variables with / *Worksheet Copy*

■ At this point you will create the formulas for the record side of the business, Record Sales and Record Expenses. You could type them in the same way that you typed the book variables, but since the record variables work almost the same as the book variables, you can save a great deal of typing by copying the book variables to create the corresponding record variables. To do so, first copy the border entries to create variable names for the record variables in the left border, then copy the body cells to create the formulas. Use another command from the Worksheet star menu, / *Worksheet Copy.

■ *Using the / *Worksheet Copy Command*

1-2-3 users, take note: this command works virtually identically with the / Copy command in 1-2-3.

The / *Worksheet Copy command asks you to specify two worksheet *ranges*. A range is a rectangular area on the worksheet, which can be as small as a single cell or as large as the full contents of the worksheet. To specify a range, you indicate two opposite corners of the rectangle you want. You

can either indicate the upper-left and lower-right corners or the upper-right and lower-left corners, in either order. To indicate the two corners of a range, move to one corner, *anchor* the cursor by pressing the Period key, and move to the other corner. Once you anchor the cursor, the highlighted area stretches as you move the cursor to the other corner. When the whole range is highlighted, press ⟵ to indicate this is the range you want.

The two ranges Javelin needs to know for this command are the range to copy *from* and the range to copy *to*. The / *Worksheet **C**opy command copies the contents of the cells in the from range and copies them into the cells in the to range. The cells in the from range are unaffected by the copy.

If the two ranges are the same size, Javelin makes one copy of the from range into the to range. If you make the to range larger, Javelin makes as many copies of the from range as will fit.

■ Copying Border Cells

You can move down and type indented entries for Volume, Sales, and Expenses, but it's easier to copy the cells in the border below the Book row into those below the Record row. Give the command / *Worksheet **C**opy and Javelin responds with the prompt *Specify range to copy from*. Note that the mode indicator in the bottom-right corner of the screen has changed from READY to POINT: Javelin wants you to point to the range. The range that you want to copy

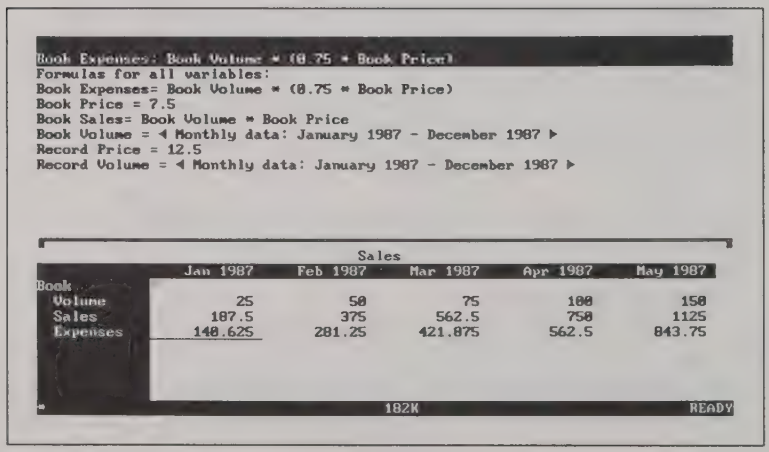

Book Expenses: Book Volume * (0.75 * Book Price)
Formulas for all variables:
Book Expenses= Book Volume * (0.75 * Book Price)
Book Price = 7.5
Book Sales= Book Volume * Book Price
Book Volume = ◄ Monthly data: January 1987 - December 1987 ►
Record Price = 12.5
Record Volume = ◄ Monthly data: January 1987 - December 1987 ►
Sales
Jan 1987 Feb 1987 Mar 1987 Apr 1987 May 1987
Book
Volume 25 50 75 100 150
Sales 187.5 375 562.5 750 1125
Expenses 140.625 281.25 421.875 562.5 843.75
182K
READY

Figure 7.9: Worksheet using indented borders

from is one column wide and three cells high, from the border cell that says Volume to the border cell that says Expenses. Move the cursor to the first cell that you want to copy from by pressing ↑ four times. Anchor the cursor here by pressing the Period key, which tells Javelin this is the first cell to copy. Javelin prompts, *Press arrows to show range.* Move the cursor down to the last cell you want to copy, the border cell containing Expenses. Javelin highlights the three cells you have selected, as in Figure 7.10.

Tell Javelin that you have finished selecting the cells to copy from, using ⟵. The cursor returns to where it was when you started the / *Worksheet **C**opy command and prompts, *Specify range to copy to.* The range you want to copy to is the three cells below the Record heading. Indicate this by moving down once.

This is the upper-left corner of the three-cell range. You don't have to indicate the other corner of the range; Javelin will assume that the to range should be the same size as the from range. Press ⟵ and Javelin copies the three border entries. Values appear to the right of Volume, since Javelin recalls Record Volume from the central information base. No values appear on the Record Sales and Record Expenses rows, because these variables are not defined yet. The screen now looks like Figure 7.11.

You can't see the last line of the copied borders because it is off the bottom of the screen. The easiest way to fix this is to switch from two screen windows to one. Give the command / **W**indow **1**, which makes the worksheet fit easily on the screen.

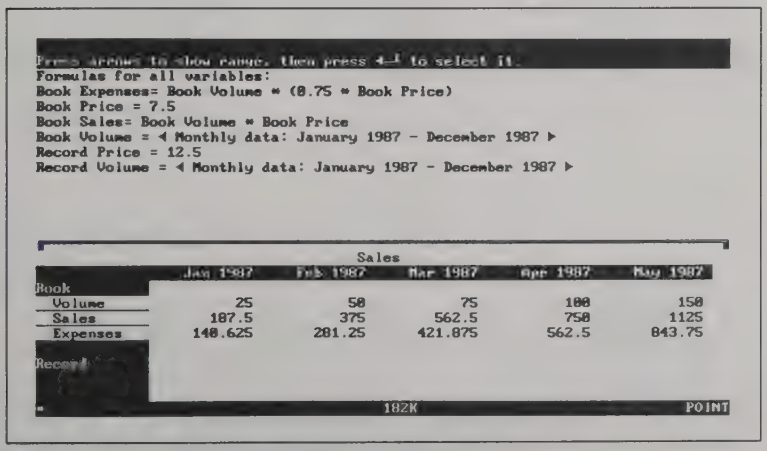


Figure 7.10: Worksheet with highlighted range to copy

■ Copying Formulas

Now you need to define the formulas for Record Sales and Record Expenses. Since their definitions are similar to those for Book Sales and Book Expenses, you can copy the formulas. When you copied the borders, you just copied label information, the names of variables. When you copy body cells, Javelin copies the formulas that define the data in the cells.

The range to copy from is the body cells displaying values of the formulas you want to copy, Book Sales and Book Expenses. You could copy the whole range that displays Book Sales and Book Expenses from January through December, but since in this case each entire row is defined by a single formula, it suffices to copy one body column, for example, the one for January.

Move to the cell displaying Book Sales for January by pressing ↑ three times and → once. Give the command / *Worksheet Copy. Anchor the cursor by pressing Period, then ↓. Move down to highlight the body cells for both of the variables whose definition you want to copy.

Press ⟵ to tell Javelin that you are done defining the range of cells you want to copy from. Now specify the beginning of the range to copy to, the first body cell for Record Sales. Press ↓ five times. You don't have to give Javelin the other corner—it will figure it out. Now press ⟵. The rows you just copied into, Record Sales and Record Expenses, display #CALC, because you have defined new formulas that have not yet been calculated. The rows copied from are unchanged.

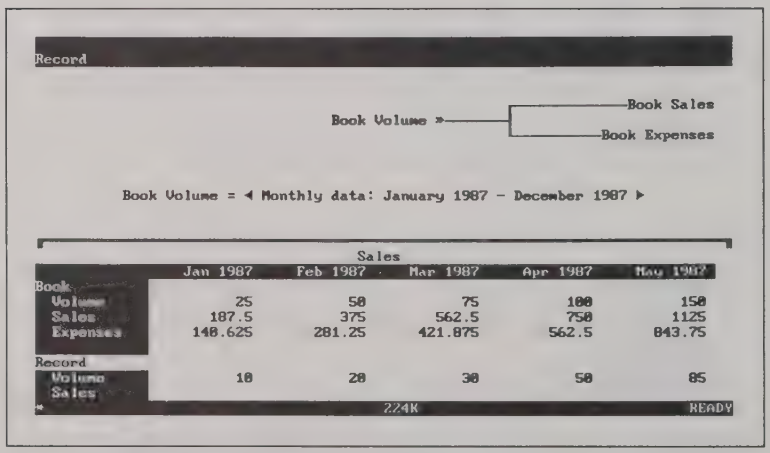


Figure 7.11: Worksheet after copying borders

Move down to look at a body cell for Record Sales. On the information line, Javelin shows its definition:

Record Sales: Record Volume * Record Price

Javelin performed a *relative copy* to create Record Sales. That is, it understood that you were copying Book Sales to Record Sales and that the difference between the source name and the target name was that the word Book was changed to Record. It then made the same change everywhere in the formula, changing Book Volume to Record Volume and Book Price to Record Price.

Relative copying is a powerful part of the Worksheet view, enabling you to create a set of variables and formulas defining one product, one region, or one department and then quickly and easily copy it to create the corresponding variables and formulas for other products, regions, or departments.

Now move down a line and look at Record Expenses. You see

Record Expenses: Record Volume * (0.75 * Record Price)

Again Javelin has copied the formula, changing every instance of Book to Record. It happens that this formula is not quite right, since the cost of records is different from that of books, and Javelin can't know that automatically. The cost factor is 0.5 rather than 0.75. This is easy to fix, using the Edit key: press F2 and Javelin puts the item the cursor was on (the formula for Record Expenses) on the edit line so you can change it. In this case you want to delete the 7, so press → until you get to the 7 and then press Del. When the formula is correct, press ⟵. Javelin accepts the corrected formula and displays it on the information line. To recalculate the model to see the correct values, press F9.

Exception Cells

■ The worksheet currently shows a useful summary of book and record information, but Book Price and Record Price are not shown. You could go to the left border of two empty rows and type in the variable names, but that would produce two entire rows of cells showing the values, as before. Since you prefer to see each value just once, use *exception cells.*

An exception cell is a cell that is not defined by the borders above and to the left of it. Exception cells can contain labels or variables. You will use exception cells to display the prices for books and records in a single cell each, rather than repeated across a whole row.

■ Entering Label Exception Cells

The first kind of exception cell you will create is a label. You have already typed labels into border cells to specify variables you wanted to see. You can type labels into body cells as well, but in Javelin 1.1, labels in body cells are treated as text only and not interpreted at all by Javelin. You can use them to include footnotes on your worksheet, draw lines, and generally spruce up the worksheet's appearance. In Javelin PLUS, you can work directly with text on your worksheets (see Chapter 20, "Strings and Dates").

To put a label for Book Price into the worksheet, move down two rows, then type the label

Book Price

The label appears in the cell, and Javelin also displays it on the information line, enclosed in apostrophes to show that it is a label. (In Javelin 1.1, the label is preceded but not followed by an apostrophe.)

■ Entering Variable Value Exception Cells

The next exception cell will display the value of Book Price. First move to the next cell to the right. Now create an exception cell that contains a value by using the command / *Worksheet **E**xception. Javelin prompts you with *Cell should show:*, to ask you to specify a value. You can type in the name of a variable.

Type the name of the variable that you want to display (or press ⟶|, the Spell key):

Book Price

The value of Book Price appears in the cell.

Create similar exception cells for Record Price, one with a label and another with the value. Move another cell to the right and enter the label:

Record Price

Move right once more. Then create the exception cell for the value, again using the command / ***Worksheet Exception**. Again type

Record Price

Javelin displays the Record Price variable in the cell, as in Figure 7.12.

■ *Highlighting Exception Cells*

In Javelin 1.1, it is not obvious which cells are exception cells, but Javelin can highlight all the exception cells on the worksheet for you. There is a setting on the Worksheet settings sheet to do this. Javelin PLUS highlights exception cells automatically, but you can use this setting to turn the highlight off.

Give the command / **D**efine **W**orksheet to display the Worksheet settings sheet. In the lower right of the settings sheet is the setting Highlight Exception Cells. Move to the setting and press ⟵, to switch between on and

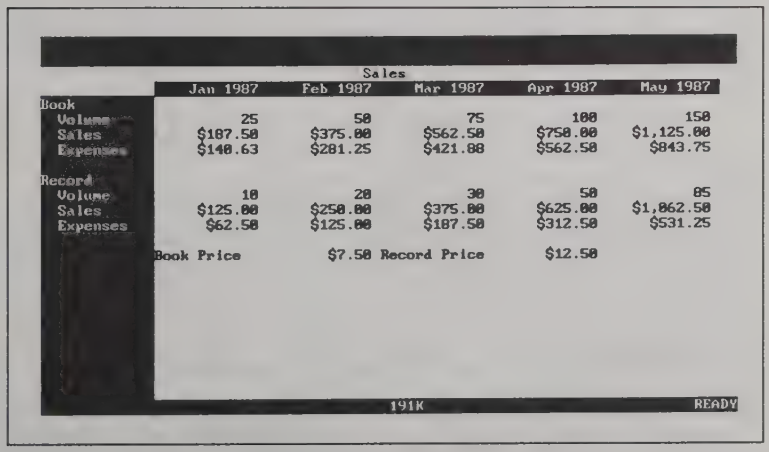


Figure 7.12: Worksheet with exception cells

off. Exit from the settings sheet by pressing Esc. The exception cells are highlighted if the setting is On.

■ Deleting and Copying Exception Cells

You can delete an exception cell by moving your cursor to the cell and pressing the Del key. You can also use the command / *Worksheet **R**ange Delete (see Chapter 13, "Advanced Worksheets and Printing").

Exception cells can also be copied using / *Worksheet **C**opy.

Creating Multiple Worksheets

■ Currently displayed on the worksheet are a group of Book variables and a group of Record variables. Each group displays an Expenses variable. Let's create another worksheet that turns the organization around to produce an Expenses heading with the various types of expenses listed below it. Javelin allows you to have as many worksheets in a model as you want, and well-organized models generally have many small worksheets.

Tell Javelin to display another worksheet and call it Expenses. You don't need to save the Sales worksheet—it is part of the model and saved to disk when you save the whole model.

Give the command / **V**iew **W**orksheet and type

Expenses

The previous worksheet vanishes and a new empty worksheet called Expenses appears. The old worksheet hasn't been deleted, just removed from view. You can come back to it any time using / **V**iew **W**orksheet.

■ Filling in Border Dates Automatically

You want to put the 12 months across the top again, but this time use an easier method to put in a lot of dates. On the Worksheet star menu, the command / *Worksheet **B**order **D**ates allows you to enter a whole range of dates into the top or left border of the worksheet.

First, press → to move to the first border cell where you want to enter dates, then give the command / *Worksheet **B**order **D**ates. Javelin now asks for the Begin Date, the first date to place in the border. It suggests the current month, which is not what you want. Type

jan87

Now Javelin asks for the End Date, the last date to place in the border. Type

dec87

Javelin now fills in the month names across the top.

In the left border, put in some variable names using indentation. Press Home, then ↓, and type

Expenses

Press ↓, then two spaces, and type

Book

In the next border cell down, type two spaces, and then type

Record

After you press ↓ once more, Javelin fills in the two rows automatically.

■ *Indented Borders and Variable Names*

The name Book indented under Expenses calls up the same variable as the name Expenses indented under Book. This is an extremely powerful aspect of Javelin. The same data defined as Book Expenses and Record Expenses in one place can be called up as Expenses Book and Expenses Record here. *The order of words in a variable name does not matter.* In a more complex model, you might have variables with names like Boston Paperback Book Promotion Expenses. That variable could appear in many differently organized presentations of your data, depending on the order of indented labels in the worksheets.

There are four more variables to be defined: Total Expenses, Overhead Expenses, Total Sales, and Profit.

■ *Entering Constant Variables into Indented Borders*

First, let's define Overhead Expenses. To put the name into the border, press a space twice and type

Overhead

Now a minor problem occurs. If you move to the right and type the value 500 into a body cell, Javelin will make the variable a monthly one with its January value being 500. Overhead Expenses isn't a monthly variable but a constant. So how you can indicate this to Javelin?

Earlier, you defined Book Price and Record Price, which are both constants, by typing the whole formula for each variable into the border cell. But now that you are using indented labels in your borders, Javelin won't let you do that.

The easiest way around this problem is to create Overhead Expenses somewhere else on the worksheet, not using an indented border. Then you can erase it from where you defined it and display it where you want it.

Define Overhead Expenses in the next row of the worksheet, by pressing ↓ and typing the whole formula into the border.

Overhead Expenses = 500

The value 500 appears across the row both in the place you just defined Overhead Expenses and in the row with the indented Overhead label, as in Figure 7.13. You can tell this is a constant because, when the cursor is on a body cell in that row, Javelin shows the value for Overhead Expenses with no date.

Just delete the extra unindented row, since you don't need it on this worksheet. The variable remains defined in the central information base, and its value is still displayed on the indented Overhead row. Press Del.

Total Expenses can easily be defined in the next indented line, so enter the name in the border and the definition in a body cell. (When you define new variables using indented border labels, you have to type the formula for the variable into body cells. Javelin will not let you type the whole formula into the border cell.) Press a space twice and type

Total

Then press → and type

= Overhead Expenses + Book Expenses + Record Expenses

(As always, in Javelin 1.1, do not type the = sign.)

Finally, create the variables Profit and Total Sales. Skip a line to separate them from the expense figures. Press ←, then ↓ twice, and type

Total Sales = Book Sales + Record Sales

Press ⟵ and type

Profit = Total Sales - Total Expenses

Press ⟵ again and your model is complete. Press the Calc key and Javelin computes all the values, as shown in Figure 7.14.

All the variables have been defined and are stored in the central information base for the model. You have also set up two worksheets, Sales and Expenses.

Converting Dates in the Worksheet

■ Let's set up one more worksheet. The Worksheet view lets you do some common date conversions without having to write any formulas. Javelin can convert between days, weeks, months, quarters, and years automatically. Just indicate the period in the borders, and Javelin does the math.

Let's assume that you need quarterly totals for Total Sales, Total Expenses, and Profit. To display a new worksheet called Quarterly, give the command / **V**iew **W**orksheet and enter

Quarterly

To get quarterly totals, fill in the names of the four quarters in the top border, that is, 1Q87, 2Q87, etc. The easiest way to do this is by using the command / *Worksheet **B**order **D**ates again.

Move into the top border and give the command. Type the begin date

1q87

and the end date

4q87

To get the totals for the year, move over to the first blank cell in the top border (four places) and type

1987

To enter the variable names in the left border, press Home, then ↓, and type

Total

Press ↓, then a space, and type

Sales

Press ↓, space, then type

Expenses

Finally, press ↓ twice and type

Profit

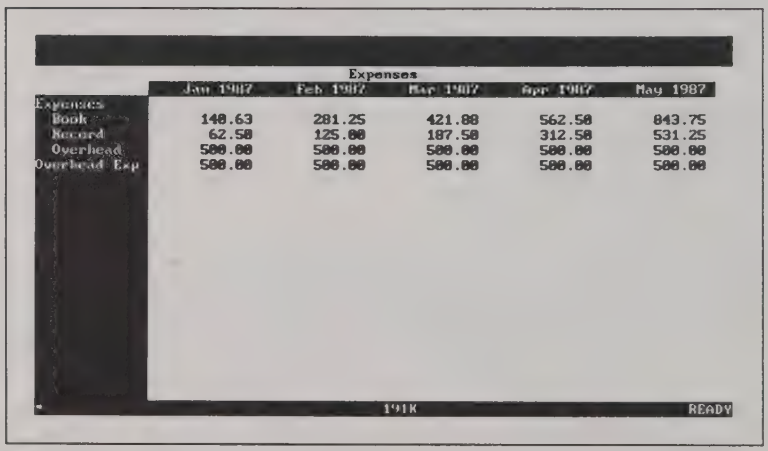


Figure 7.13: Worksheet with expense information

As you type in the variable names, the values appear, with the monthly data summed to get quarterly and yearly results (see Figure 7.15). You can also intersperse monthly, quarterly, and yearly data, such as a report with columns (or rows) for January, February, March, 1Q, April, May, June, 2Q, and throughout the year. In this example, Javelin has summed up the values to combine smaller periods into larger ones. Javelin can also use other combining methods, such as averaging the values when combining percentages (see Chapter 13, "Advanced Worksheets and Printing").

Formatting Variables, Worksheets, Rows, Columns, and Cells

■ Before you finish looking at worksheets, change the formats in which the numbers are displayed, to make the worksheets suitable for printing.

■ *Number Formats*

Javelin has many different number formats:

- In Automatic, Javelin automatically uses as many digits as it needs to display the value of the number;

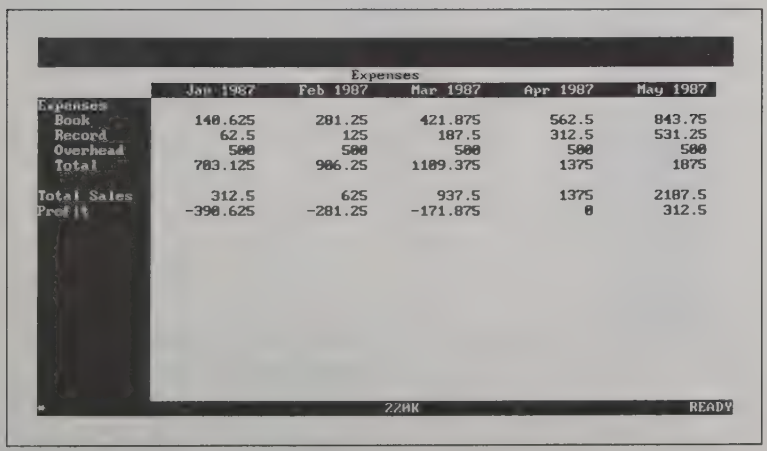


Figure 7.14: Complete Expenses worksheet

- In Fixed, you tell it how many places to display to the right of the decimal point (for example, 1234.56 is displayed as 1234.5 if you specify one decimal place);
- In Percent, it displays a number as a percentage, with a percent sign (for example, .12 is displayed as 12%);
- In Commas, it acts like Fixed except that it puts commas between every group of three digits and negative numbers are shown in parentheses (such as 1,234.56);
- In Dollars, Yen, Franc, Swiss Franc, Deutsch Mark, and Guilder, it acts like Commas except it also displays the appropriate currency symbol (as in $1,234.56).

Except in Automatic format, you specify the number of digits to display to the right of the decimal point, from no digits (whole numbers) to 12. For most of the variables in our model, the format to use is Dollars with two decimal places, for dollars and cents. For Book Volume and Record Volume, Commas format with no decimal places is better.

Javelin has five different settings sheets where you can specify the formats to use when displaying variables' values.

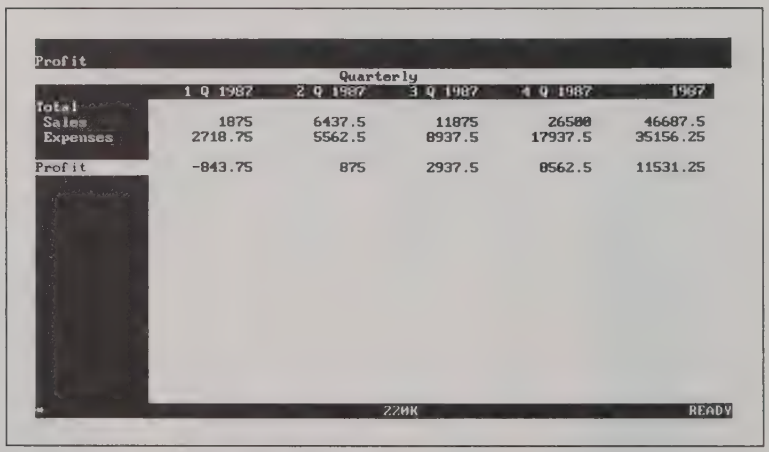


Figure 7.15: Quarterly and yearly totals

- Model Settings Sheet: This specifies the format to use everywhere in the model, except where other formats override it. Usually, most variables in a model are formatted the same, so by setting the model format you can format them all. Display this settings sheet with / **D**efine **S**ettings.
- Variable Settings Sheet: When a variable needs a format different than the standard one for the model, you can set the format for the variable specifically. Display this settings sheet with / **D**efine **V**ariable.
- Worksheet Settings Sheet: You can set a format to use for a worksheet, overriding any formats set for the model or for variables that appear on the worksheet. Display this settings sheet with / **D**efine **W**orksheet.
- Worksheet Borders: You can set a format for any worksheet border cell, whether top, side, or corner, that overrides the worksheet format for the cells controlled by that border. Display this settings sheet with / *Worksheet **F**ormat, placing the cursor in a worksheet border cell.
- Worksheet Cells: You can set a format for an individual worksheet body cell. Display this settings sheet with / *Worksheet **F**ormat, placing the cursor in a worksheet body cell.

This probably seems complex, and it is. For most applications, setting formats on the Model and Variable settings sheets is quite adequate. Worksheet, border, and cell formatting are useful when you are creating a worksheet to produce a report in a very specific format.

■ *Setting the Model Format*

To put this into practice, first set the model format on the Model settings sheet. Display the settings sheet with the command / **D**efine **S**ettings. You want to format the numbers in the model as dollars, so move the cursor to the right, and down five times so that it is on the line that says Dollar. Set this format by pressing ⟵, so that it looks like Figure 7.16. The number of decimal places is set to two (on the top right of the settings sheet), which is Javelin's guess for the number of decimal places that you want. The guess is right, so don't do anything else here.

Press Esc. The settings sheet disappears, and on the worksheet all values are now displayed as dollars. This is enough for the Quarterly worksheet, since all the variables represent dollar values.

■ *Setting a Variable's Format*

Now check the formats on the Sales worksheet. (You may want to use the Spell key rather than typing worksheet and variable names in the commands that follow.) To look at that worksheet again, give the command / **V**iew **W**orksheet and type

Sales

The Sales worksheet reappears. All variables are formatted as dollars, which is correct for all except Book Volume and Record Volume. You can fix this by using the Variable settings sheet. Use the command / **D**efine **V**ariable to display the settings sheet for Book Volume. When Javelin prompts for a variable name, type

Book Volume

Notice that on this list of format settings is an additional choice—Default indicates that you want this variable to use the default format for the model, that is, the format you chose on the Model settings sheet.

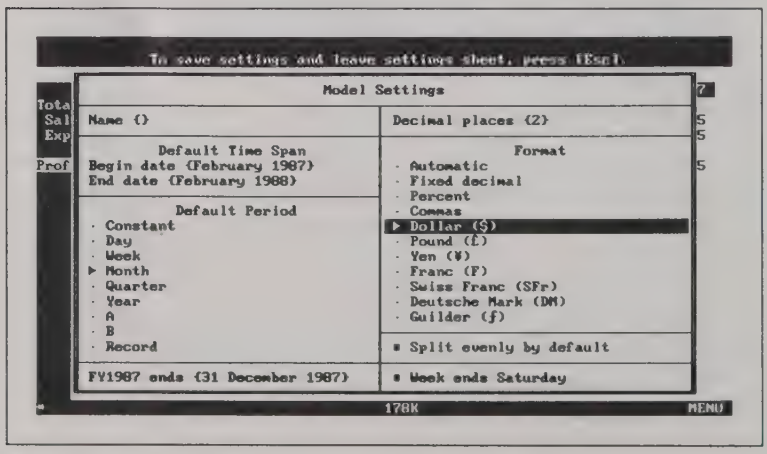


Figure 7.16: Setting a format on the Model settings sheet

You want to set the variable to Commas format, so move the cursor down to the line Commas and select it. The number of decimal places is still two, but you want it to be zero, since you don't expect to sell fractional numbers of books. Move up to the Decimal Places line and set the number of decimal places by typing

```
0
```

The format is set correctly, as in Figure 7.17.

Now you're done, so leave the settings sheet by pressing Esc. The settings sheet disappears, and the values for Book Volume on the worksheet now appear as whole numbers without dollar signs.

To change the format for Record Volume with the same result, you'll use a slightly different method of specifying the variable to format. Move the cursor to a body cell that displays Record Volume. Then display the Variable settings sheet, using the command / **D**efine **V**ariable.

This time Javelin prompts with *Variable (Record Volume).* Since the cursor was in a cell that displays a variable, Javelin guesses that you want to change that variable. The guess is correct, so press ↵ to confirm that you want to work on Record Volume.

Now the settings sheet appears. Set the format exactly the same way that you did for Book Volume and exit from the settings sheet. Record Volume now appears as whole numbers, so you're done formatting numbers on this worksheet.

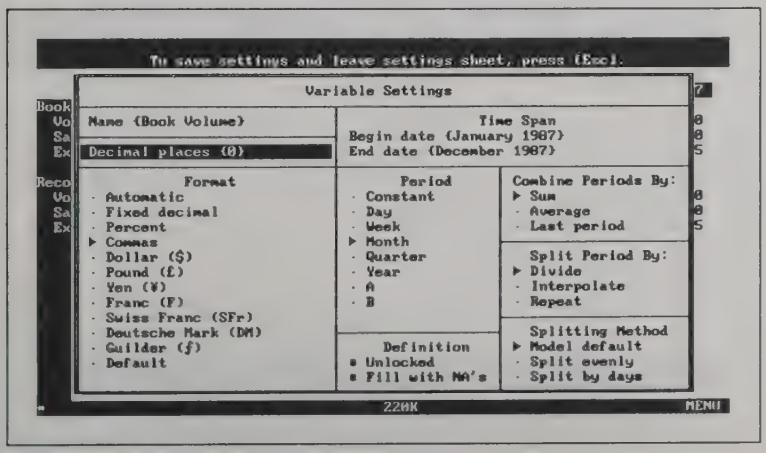


Figure 7.17: Setting a format on the Variable settings sheet

■ *Setting Worksheet, Border, and Cell Formats*

Formatting a worksheet is similar to the other kinds of formatting. Since the Expenses worksheet contains only money numbers, you want to format it without dollar signs except in the Profit row.

You set the format for a worksheet on the Worksheet settings sheet, so display the settings sheet for the Expenses worksheet with the command / **D**efine **W**orksheet. You don't have to have the worksheet displayed to see its settings sheet. Now type

Expenses

The settings sheet appears as in Figure 7.18.

Move down and select the Commas format, then leave the settings sheet by pressing Esc.

When you display the worksheet, you'll see that all the numbers are displayed without dollar signs. Give the command / **V**iew **W**orksheet and type

Expenses

Everything is right, except that you want to format the Profit line with dollar signs. To set the format for a row, move to the border cell that controls the row and use the command / *Worksheet **F**ormat. The line you want is the last one on the worksheet, so move to it directly using the End key.

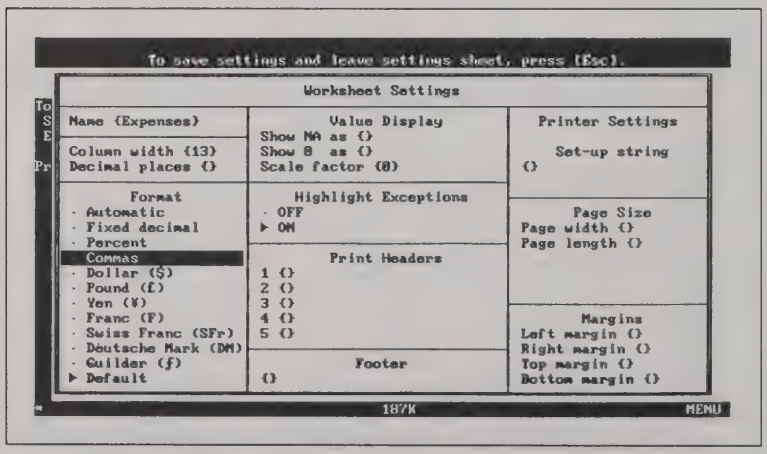


Figure 7.18: Setting a format on the Worksheet settings sheet

When you press End, the information line says, *End: Press arrow key to select direction.* Press the arrow key for the direction you want to move. Javelin moves the cursor to the end of the worksheet in the direction you indicated. That is, it moves to the last row or column in that direction that contains anything.

To move to the bottom end of the worksheet, press End and ↓. The cursor moves down to the last row that has something in it, which is the Profit row. Use / *Worksheet **F**ormat to display the Format settings sheet, as in Figure 7.19.

Move the cursor down to Dollar and select it, and exit from the settings sheet using Esc. The row on this worksheet is reformatted with dollar signs: this doesn't affect the way the Profit variable is displayed on any other worksheet or in any other view.

You can format columns or individual cells in exactly the same way, by moving the cursor to the border cell that controls the column you want to format, or to the individual body cell that you want to format, and then using / *Worksheet **F**ormat to display the Format settings sheet.

As you've seen, the format set on the Model settings sheet is the default for all the numbers in the model. This can be overridden for individual variables by using the Variable settings sheet, or for a whole worksheet by using the Worksheet settings sheet. Within a worksheet, you can control the format of rows, columns, and individual cells using the Format settings sheet, with the cursor position in the worksheet indicating what you want to format.

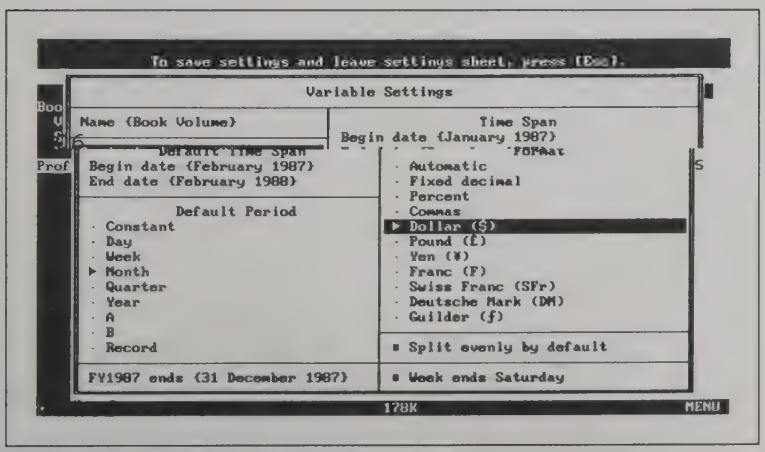


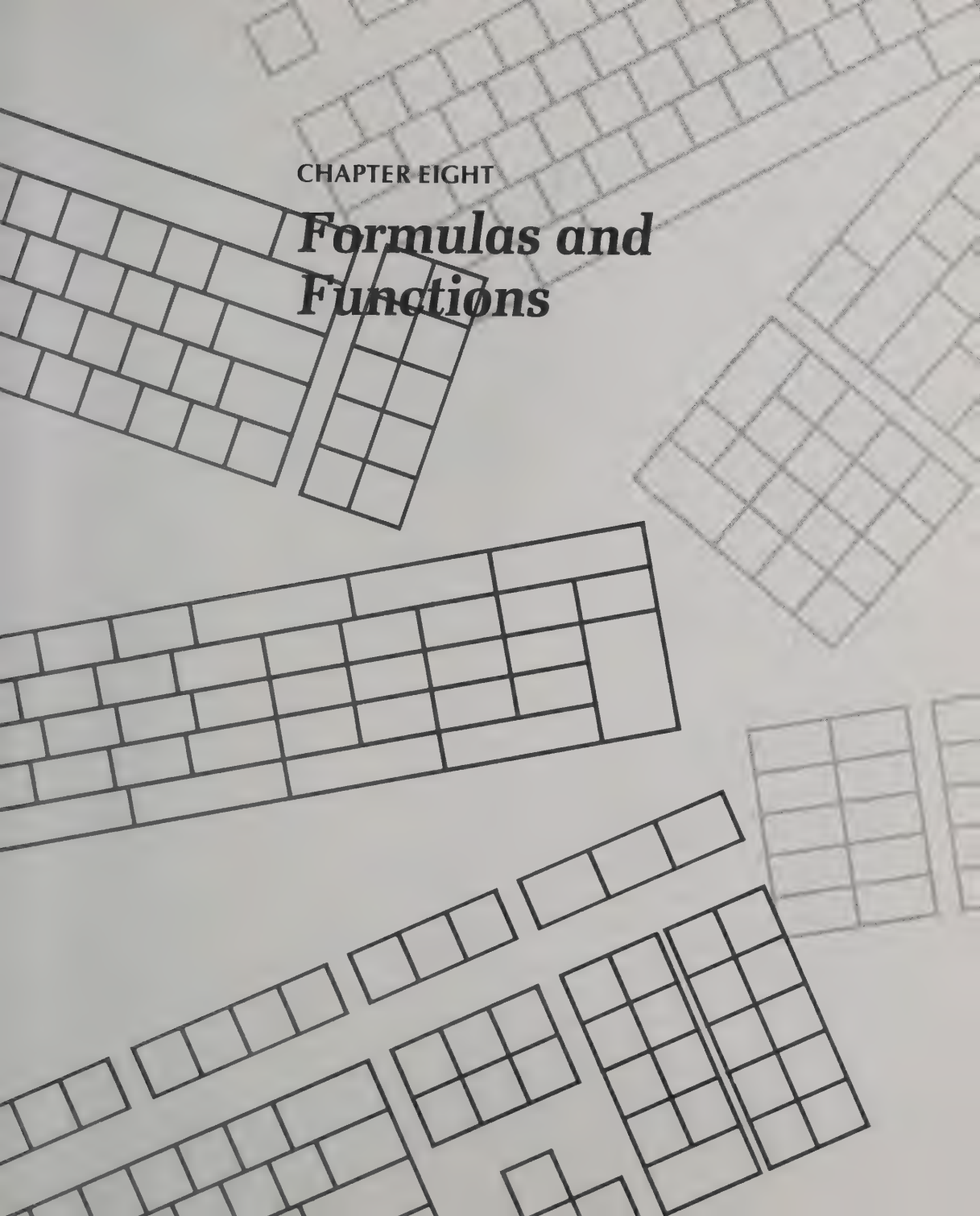
Figure 7.19: Setting a border or cell format

Save this worksheet as usual: give the command / **F**ile **S**ave and use the file name of your choice.

Conclusion

■ The Worksheet view can be used for creating variables and entering data as well as creating formatted reports. We'll discuss the rest of the choices on the Worksheet star menu, as well as other worksheet topics, in Chapter 13, "Advanced Worksheets and Printing."

But first, let's look at Javelin's formulas to see how to create more complex variables. We'll also look at Javelin's many built-in functions.



CHAPTER EIGHT

Formulas and Functions

At the heart of any model are the formulas that define the relationships among its variables. So far our formulas have been limited to simple arithmetic. In real models your formulas will relate your figures in far more complicated ways. For example, you may need to know the highest value that a variable has reached to date—the maximum temperature, maximum weekly sales, or the highest stock price of this year. This sort of calculation requires a more complex formula. Javelin has many preprogrammed *functions* to help you do complex calculations, such as absolute value, minimum value, maximum value, statistical analysis, depreciation, and net present value.

In addition, there are situations in which you need to calculate a variable differently under different conditions. For example, the book selling price may be 1.5 times the book cost if it costs up to $20, and 1.35 times the cost if the book costs more than that. Javelin's logical functions can be used to do this type of conditional calculation.

In this chapter we will cover several aspects of Javelin that relate to writing more complex formulas. We will describe in detail the different ways you can define a variable, and we'll talk about formulas in Javelin, focusing mainly on the built-in functions. Along the way, we will also digress into the numeric formats that determine how Javelin displays your numeric data.

The Sample Model: ECON

■ Our example in this chapter will be a simple model of economic statistics about the United States. The model, which is called ECON, contains data about the U.S. economy during the period 1966 through 1985. The data come from the *Economic Report of the President, February 1986.* The model includes variables for gross national product (GNP), federal defense spending, federal nondefense spending, unemployment, imports, and exports. All variables in the model are annual. (Federal nondefense spending does not reflect federal revenue-sharing funds given to states.)

There are seven input variables for this model. GNP, Federal Defense Spending, Federal NonDefense Spending, Imports, and Exports are all entered in billions of U.S. dollars. The unemployment rate is expressed as a percentage of the labor force. The GNP Deflator is a measure of inflation that allows conversion of dollar values for previous years into 1982 dollars. Table 8.1 lists the values for each of the six input variables.

Using / Define Variable to Create Variables

■ You have created variables in three ways: by giving the command / View Table for the new variable; by typing the variable name in the border

	GNP		FEDERAL SPENDING				UNEMPLOYMENT%
	Billions of $	Deflator	Defense	Non-Defense	Imports	Exports	
1966	772.0	35.0	62.0	18.3	39.1	46.6	.037
1967	816.4	35.9	73.4	19.3	42.1	49.5	.037
1968	892.7	37.7	79.1	21.0	49.3	54.8	.035
1969	963.9	39.8	78.9	21.1	54.7	60.4	.034
1970	1015.5	42.0	76.8	22.0	60.5	68.9	.048
1971	1102.7	44.4	74.1	25.8	66.1	72.4	.058
1972	1212.8	46.5	77.4	28.4	78.2	81.4	.055
1973	1359.3	49.5	77.5	28.9	97.3	114.1	.048
1974	1472.8	54.0	82.6	33.6	135.2	151.5	.055
1975	1598.4	59.3	89.6	39.6	130.3	161.3	.083
1976	1782.8	63.1	93.4	42.9	158.9	177.7	.076
1977	1990.5	67.3	100.9	50.3	189.7	191.6	.069
1978	2249.7	72.2	108.9	52.9	223.4	227.5	.060
1979	2508.2	78.6	121.9	56.1	272.5	291.2	.058
1980	2735.0	85.7	142.7	65.4	318.9	351.0	.070
1981	3052.6	94.0	167.5	74.8	348.9	382.8	.075
1982	3166.0	100.0	193.8	78.9	335.6	361.9	.095
1983	3401.6	103.8	215.7	69.2	359.4	354.1	.095
1984	3774.7	108.1	237.0	76.0	443.8	384.6	.074
1985	3992.5	111.7	262.0	91.9	444.8	370.4	.071

Table 8.1: Input variables for ECON model

and the data in the body of a worksheet; and in several views, by typing the formula that described the variable.

No matter which way you define it, the first time you refer to a variable you should use the word order you will use most often and the capitalization you prefer. Javelin will use this word order and capitalization whenever you ask it to supply the variable name for you, such as when using the Spell key. Later on, when you use the variable, word order in variable names does not matter. If you name a variable Domestic Wholesale Price, Javelin is willing to let you refer to it as Wholesale Price Domestic or as Price Domestic Wholesale.

There is a fourth way of defining a variable, by using the command / **D**efine **V**ariable. This displays the Variable settings sheet to allow you to define the settings that control how the variable will be formatted and its period. We will use / **D**efine **V**ariable to enter the economic database.

■ *Creating the ECON Model*

Here's how to create this model, starting with a blank model.

1. Set the Model settings sheet (explained in Chapter 4) to tell it the types of data you will enter. The command / **D**efine **S**ettings displays the Model settings sheet.

Make the following changes:

- Set the default period for variables in the model to Year.
- Set the default time span for variables in the model to start at 1966 and end at 1985.
- Set the default format for numbers to Commas with one decimal place.

Leave the settings sheet by pressing Esc. Now when you create each variable in the Table view it will correctly display the years 1966–1985 for data entry, and the values will be displayed with commas.

2. Create the first variable, GNP, using / **D**efine **V**ariable GNP. The Variable settings sheet appears. All the settings should be correct—it will use the ones you set for the whole model. For variables not in this format (one decimal place), you can set the format you want here. Leave the settings sheet by pressing Esc.

3. Enter the data from Table 8.1 for the variable in the Table view. The easiest way to do this is to use the Num Lock key to switch the numeric

keypad into its "numbers" meaning. Enter each number on the keypad and press ⏎. The cursor moves down automatically to the next period.

4. Repeat steps 2 and 3 for each input variable.

5. Save the model as ECON before continuing with the chapter; in case you do something wrong, you won't have to enter the data again.

■ *Using the Variable Settings Sheet*

You have used the Variable settings sheet several times by now—let's take a closer look at the settings on this sheet and the information it gives about your variable. Unlike the Model settings sheet (which affects the whole model) the settings on the Variable settings sheet affect only the variable you are defining; for the most part, this allows you to override the settings in the Model settings sheet.

Give the command / **D**efine **V**ariable and type the variable name

GNP

You will see the Variable settings sheet for GNP. Table 8.2 describes what each area of the settings sheet does.

If you create a variable without using the Variable settings sheet, for example, by entering a formula for it in the Formulas or Worksheet view, the variable takes its settings from the Model settings sheet. After you have created the variable, you may go at any time to the Variable settings sheet and change the settings.

Typing in Numbers

■ So far, you have typed in numbers in a fairly straightforward way. Now it's time to learn exactly what Javelin will and will not accept as numbers. Furthermore, if you are using Javelin PLUS, you will want to know how Javelin distinguishes between numbers, dates, and text.

To begin with numbers, as you have seen, you can type simple numbers as a series of digits (123), digits with a sign (−123 or +123), or digits with a decimal point (123.45, −123.45, or +123.45). You cannot use punctuation, such as commas, percent signs, dollar signs, *E*'s (for exponentiation), or

Setting	Purpose
Name	If you change this, the variable name is changed throughout the model. However, if the variable appears in the border of a worksheet, that worksheet border is not changed. When you change the name of a variable, review your worksheets to make sure none of them is affected. Also see the / *Worksheet Name change command in Chapter 7.
Decimal Places	Number of places displayed to the right of the decimal point. If you don't fill in any value here, the number of decimal places displayed depends on the format you choose (see Table 8.3).
Format	Display variable in this format. See Table 8.3 for a list of formats. The Default choice means the variable will be formatted using a format set on the Model settings sheet.
Time Span	Begin and end dates for the variable. Unlike in the Model settings sheet, you cannot enter values here. For an input variable, the time span is determined by the data you type in. For a calculated variable, it is determined by the time spans of the variables and functions upon which this variable depends.
Period	Period of the variable, that is, how often the values for this variable change (daily, weekly, monthly, quarterly, yearly, or a custom period). If you change the period of an input variable after you have entered data for it, Javelin deletes the values you entered. For a calculated variable, the formula determines the period of the variable.
Locked/ Unlocked	You cannot change the values or formula of a locked variable. This can help you avoid unintentionally changing parts of your model. To change a locked variable, toggle this setting. Since you (or anyone else) using your model can do this at any time, this is only an aid to keep you from making accidental changes.
Fill with Blanks/ Fill with NA's	These are the two options for dealing with missing values in input variables. Javelin leaves blanks alone (Fill with Blanks) or with the special value Not Available (Fill with NA's). In Javelin 1.1, Fill with Blanks is Fill with 0's.

Table 8.2: Settings on the Variable settings sheet

Setting	Purpose
Combine Periods By	These are the methods you can specify for how Javelin combines smaller periods into larger ones when doing time conversions. See Chapter 9, "Javelin and Time."
Split Period By	These are the methods you can specify for how Javelin splits larger periods into smaller ones when doing time conversions. See Chapter 9.
Splitting Methods	Smaller periods do not divide larger periods evenly: not all months have the same number of days. The splitting method gives Javelin some options for handling this. See Chapter 9.

Table 8.2: Settings on the Variable settings sheet (continued)

parentheses (for negative values) in numbers, even though at times Javelin uses all these when displaying numbers. This punctuation is part of numeric formatting, controlling how Javelin shows numbers to you, not how you enter numbers to Javelin.

In addition, almost anywhere you can type in a number to Javelin, you can enter a formula instead. Your formulas can be as simple as adding two numbers (5.25+2.38) or as complicated as you want, using parentheses ((1.2*(153+78+43))*(2 ^ (1–.07))). Most often, though, you will find yourself using formulas to enter fractions. For instance, to enter a stock price of 25 and seven-eighths, you could type 25+7/8 or 25.875. Javelin 1.1 replaces the formula with its answer as soon as you hit ⟵; unlike Lotus 1-2-3, the formula does not remain. In Javelin PLUS, you can use any of the built-in numeric functions as well as the mathematical operators.

Javelin PLUS allows you to type text and dates as values for variables. This means that if you type in a formula for a number, *before* you strike the ⟵ key you must press the Calc key (F9) to tell Javelin to compute the answer. If you do not, Javelin PLUS will save the formula as text, rather than calculating it as a number. If you forget to do this, you can edit the formula using the Edit key (F2), remove the quotes that Javelin PLUS puts around text, press the Calc key, and press ⟵.

Typing in dates is described in Chapter 9, "Javelin and Time." Typing in text and dates as values for variables is described in Chapter 20, "Strings and Dates."

Numeric Formats

■ *Numeric formats* specify how the values for a variable should be displayed. The format you select controls how Javelin displays numbers in the Table view, the Worksheet view, and the upper-left corner of the Quickgraph view. In other views, particularly the Chart view and the Graph view, Javelin displays numbers very simply, without punctuation.

Several settings control what format is used. In Chapter 7, you used all four settings sheets that contain format settings: Model, Variable, Worksheet, and Format. The Format sections of the Variable and Model settings sheets are shown in Figure 8.1. The additional selection, Default on the Variable settings sheet, indicates that this variable should be displayed in whatever format you have chosen for the model.

A numeric format has two parts: the number of decimal places to be shown and the name of the format to be used. The default number of decimal places for all formats is two (it displays two digits to the right of the decimal point, even if they are zero), except for the Automatic format. When you select Automatic, it does not allow you to enter anything under Decimal Places in the settings sheet.

If a number has more digits than can be displayed to the right of the decimal point, the number is rounded. That is, if you specify two decimal places,

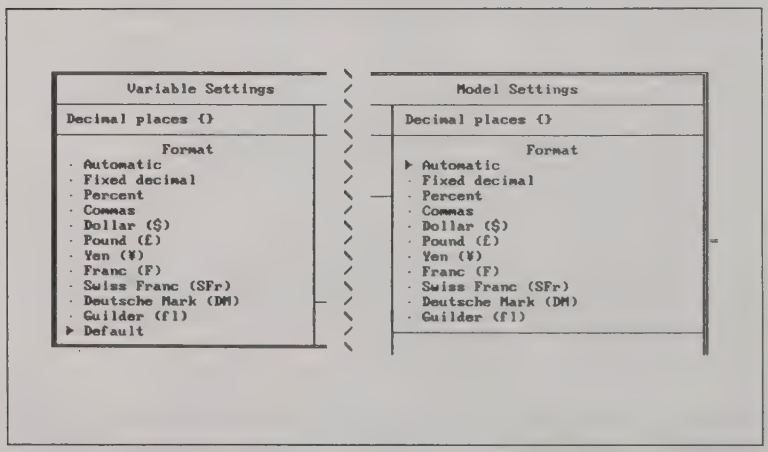


Figure 8.1: Format sections of Variable and Model settings sheets

56.784 is displayed as 56.78 while 56.785 is displayed as 56.79. Calculations are always performed using all the digits Javelin knows about. The Decimal Places setting affects only how the numbers appear. The formats, descriptions of them, and examples of what each displays are given in Table 8.3.

Format	**Description**	**Examples**
Automatic	Ignores the decimal places setting and uses as many digits as necessary to show the number. No entry is allowed under Decimal Points in the settings sheet.	1234 56.7890 −55
Fixed	Always displays the number of digits to the right of the decimal point as specified under Decimal Places.	1234.00 56.58 −55.00
Percent	Numbers are multiplied by 100 and displayed with a percent sign after them. As with rounding, Javelin remembers the number as you typed it in; the format affects only the display. For example, if you enter .1234, it will display as 0.1234 in Automatic format, but as 12.34% in Percent format (with two decimal places).	123.45% 1.00% −12.30%
Commas	Commas are inserted to separate each group of three digits to the left of the decimal point. Negative numbers are displayed in parentheses. See later in this chapter for international variations on this format.	1,234.56 (55.00) (5,678.90)
Currencies: Dollar ($) Pound (£) Yen (¥) Franc (F) Swiss Franc (S Fr) Deutsch Mark (DM) Guilder (ƒ)	These seven currencies come predefined in Javelin; each has a name (listed at left), a symbol (indicated in parentheses on the settings sheet), and some formatting information (see text under international variations). Generally, currency formats (like Dollar) are just like Commas format, but to the left of the numbers, they display a currency symbol.	$1,234.56 ($55.00) ($5,678.90)

Table 8.3: Numeric formats in Javelin

■ International Variations on Numeric Formats

Javelin 1.0 allows you to choose whether you want to use the U.S. decimal-point and comma system (e.g., 1,234.56) or the European system (1.234,56). Choose one from the Configuration settings sheet. Your choice affects the Commas format and the currency formats throughout the model.

Javelin 1.1 and PLUS allow more flexibility. First, instead of choosing U.S. or European commas and decimal points, you have four choices on the Configuration settings sheet, as shown in the examples below (see Figure 8.2):

- 12,345.67
- 12 345.67
- 12.345,67
- 12 345,67

In addition, you have flexibility in defining currency formats. In Javelin 1.1 and later, you should think of currency formats as a way of defining the *units* of your variables, not just currencies. For example, you might have variables that measure in nonmonetary units such as metric tons, pounds per square inch, or barrels. This capability is particularly useful when you are mixing unit values or numbers of different scales (thousands, millions, etc.) in the same model.

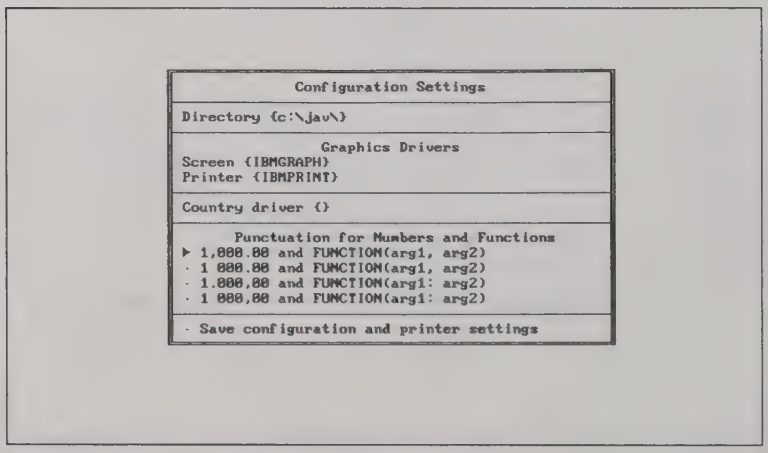


Figure 8.2: Javelin PLUS Configuration settings sheet

The eight currencies in the Format parts of the settings sheets are predefined units. The Number Display settings sheet (which was new in Javelin 1.1) controls the four elements that make up a currency. Each currency has a name, a symbol, a location where the symbol should appear in a currency figure (called the *placement*), and a form for displaying negative values. In addition, in Javelin PLUS you may choose a form for displaying negative values in the standard formats: automatic, fixed decimal, percent, and comma. Figure 8.3 shows the Javelin PLUS Number Display settings sheet, which you display by using the command / **D**efine **N**umber. Table 8.4 describes the settings on the Number Display settings sheet. Use of punctuation (commas, spaces, and periods) is controlled by the command / **D**efine **C**onfiguration settings sheet.

The Point Key

■ One important thing you can do to make your models clear is to give your variables descriptive names. But this can make entering new formulas tedious; that is why Javelin has the Spell key (⟶|). There is another way Javelin helps you with long names: F4, the Point key, which is useful if the variable name you want appears somewhere in a view that is on the screen.

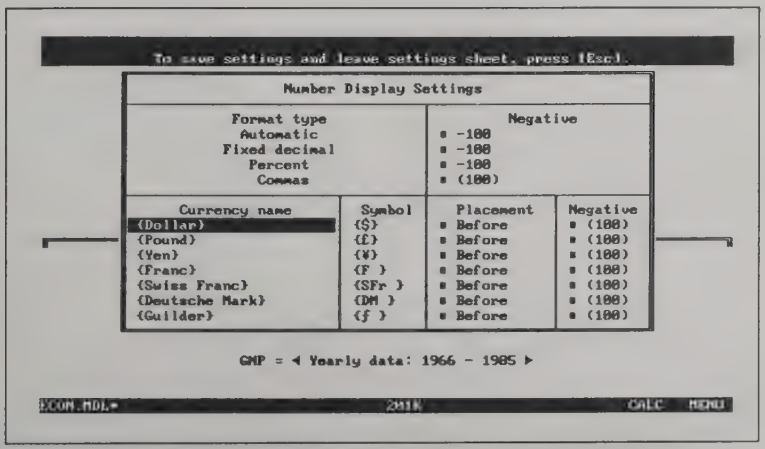


Figure 8.3: The Number Display settings sheet

Any time you have to enter a variable name you can point to it, rather than typing its name, by using F4.

Once you've pressed the Point key, you can move the cursor until you come to the variable name you want. If two windows are displayed, you can switch to the other view by using the Window key (F6). If the variable you want is in the view but not on the screen (for example, on the list of formulas in a Formulas view but off the bottom of the screen), the view will scroll until the variable comes into sight.

As you move your cursor around the screen, Javelin displays on the edit line the name of the last variable name you pointed to. When you come to

Setting	Purpose	Possible Values	Sample Display
Format Type	One of Javelin's standard formats (Javelin PLUS only).	Automatic Fixed Decimal Percent Comma	Table 8.3
Negative (for Format Types)	Determines how negative numbers are displayed: with a minus sign or in parentheses, for each format type (Javelin PLUS only).	(100) −100	−12345 −123.45 −5.1% (6,123.45)
Currency Name	Name that appears in Format part of settings sheets.	Any name up to 15 letters.	
Symbol	Symbol representing the currency or other units you use, including letters or special characters.	$ £ ¥ *f* MT (Metric Tons) K (Thousand) PSI (Pounds per Square Inch) MPG (Miles Per Gallon) Note: may include spaces.	$1,123.50 *f*325,120 500MT 59K 248 PSI
Placement	Where the symbol should appear in the number.	Before After Middle (at decimal point) Far Left	$1,123.50 500MT 538Bf22 SFr 5,724
Negative (for Currency)	Determines how negative numbers are displayed: with a minus sign or in parentheses, for each currency.	(100) −100	$−1,123.50 (500MT) (538Bf22) SFr −5,724

Table 8.4: Settings on the Number Display settings sheet

the variable you want, signal Javelin that you have found it by pressing ⟵.

If are using the Point key to enter a formula and the formula continues after this variable name, instead of pressing ⟵, press the next character in the formula. Javelin displays the variable name you pointed to on the edit line, enters the next character you typed, and waits for you to continue entering the formula.

If you want to cancel the pointing operation, press Esc. When you are in the process of pointing, this does three things: it returns the cursor to your original place on the screen, removes the name of what you were pointing at from the edit line, and puts you back into Edit mode, where you can change the information on the edit line. (Remember that when you edit information on the edit line, pressing Esc tells Javelin to ignore everything you have typed so far on the edit line. So use it with care while pointing.) Table 8.5 shows key functions in Point mode.

Try using the Point key to enter a variable name when giving a command. First, make a worksheet that has your input variables on it, so that you can check your data against the listing in Table 8.1. Use the command / **V**iew **W**orksheet to create a worksheet called Input Variables.

Move the cursor to the first cell in the top border of the worksheet and enter the names of two variables. Type

GNP

press →, and type

Imports

Key	Description
⟵	Ends both Point mode and Edit mode, storing your completed formula.
Point	Ends Point mode, but you remain in Edit mode, editing your formula.
F2	Same as Point key.
Cursor keys	Move the cursor. If the cursor highlights a variable, display that variable name on the edit line.
Any other key	Types characters on the edit line.

Table 8.5: Key functions in Point mode

Fill the left border with dates for years. In the first border cell in the left border (not the corner cell), give the command / *Worksheet **B**orders **D**ates, with a begin date of 1966 and an end date of 1985. The numbers appear for GNP and Imports.

Let's use the Point key to display the Variable settings sheet for one of these variables. Give the command / **D**efine **V**ariable. When Javelin prompts you for the name of a variable, press F4.

The POINT indicator appears on the status line. When you move the cursor, Javelin displays the name of the variable you are currently pointing to on the edit line. In the Worksheet view, this is the variable defined by the top and left borders. Move the cursor around the window, and variable names appear on the edit line.

Move the cursor to GNP and press ⟵. You will see the Variable settings sheet for GNP. Make sure you like the format and decimal places for the variable, then press Esc.

You could also have used the Spell key (⟶|) to choose a variable name rather than typing it, but the Spell key displays a list of all the possibilities. The Point key is more convenient when you have a lot of variables and the one you want is already on the screen.

You can also use the Point key when creating a formula. Make a new variable, Adjusted GNP, which uses GNP Deflator to adjust the GNP figures to 1982 dollars. To do this, you divide the actual GNP dollar figures by the deflator and then multiply by 100.

Adjusted GNP = GNP / GNP Deflator * 100

To enter the variable, move the cursor to the border cell of an empty column in the Worksheet view. Enter the name of the new variable by typing

Adjusted GNP

Move the cursor down into the data area to enter the formula. The formula starts with a variable name that is already on the screen, so use the Point key: press F4. This tells Javelin that you want to enter a formula and point to a variable name. The variable name Adjusted GNP appears on the edit line, because that is the variable displayed in the column the cursor is in.

Move the cursor left to the GNP column. The variable name GNP is displayed on the edit line, which is the variable you want to use in the formula.

To continue writing the formula, don't press ⟵. Press the next character you want in the formula: /. The formula now reads

GNP/

Use the Spell key to enter the variable name GNP Deflator into the formula: press ⟶|. The list of possibilities presented by the Spell key starts with the variable names in the model, followed by a list of Javelin functions (there are too many to fit on the screen at once).

Select GNP Deflator. Now the formula reads

GNP/GNP Deflator

Type the rest of the formula.

***100**

Press ⟵ to finish, and recalculate the model by pressing F9.

Order of Precedence

■ As your formulas get more complex, it is important that you know the order in which Javelin does the calculations. The formula

a + (b * c)

gives a different answer than

(a + b) * c

Table 8.6 shows the *order of precedence* that Javelin uses when it calculates a formula. You can use parentheses to override this order; Javelin does calculations that are in parentheses before doing other operations in a formula. In fact, you should use parentheses whenever the order of precedence is not absolutely clear or whenever your formula is complex.

What Is a Function?

■ Sooner or later you will want to do something more than simple arithmetic in Javelin. That's when you need functions, which let you combine or transform one or more numbers in interesting ways that go beyond arithmetic. To cite three examples, there is a function that chops fractions off numbers (INT), a function that computes bond prices (BONDPRICE), and a function that tells you the number of days between two dates (BETWEEN).

All functions have this format:

name(arguments)

The *name* of the function is a single word or an abbreviation, such as INT (short for *integer*), BONDPRICE, or BETWEEN. The *arguments* are the pieces of information the function needs to give the result you want. In the examples above, the arguments are the value you want to take the integer value of, the information necessary to compute a bond price, or the dates you are interested in.

Some functions (like INT) require one argument, some (like BONDPRICE and BETWEEN) need several arguments to calculate their answer, and others (like SUM) accept an unlimited number of arguments. If your function

Symbol	Meaning	Sample Formula	Javelin Result
@	Value of a variable at a particular date; see Chapter 9, "Javelin and Time."	A@1Jan86	A@1Jan86
–	Unitary minus (multiplies a number by -1).	–A@1Jan86	–(A@1Jan86)
^	Exponentiation (x to the power of y).	–B ^ 5	(–B) ^ 5
* /	Multiplication and division.	C ^ 2/6	(C ^ 2)/6
+ –	Addition and subtraction.	2+3*4	2+(3*4)
< > <= >=	Logical operations; see "Logical Function IF" in Table 8.7.	A>B+1	A>(B+1)
;	"Or else"; see Chapter 9.	A+B;C	(A+B);C

Table 8.6: Mathematical operators and order of precedence

takes more than one argument, the arguments are separated by commas. (International users should note that when you change to the European comma/decimal-point system, which uses a comma before the fractional part of a number, you must use colons to separate function arguments.)

Functions also distinguish between two different kinds of arguments: time series and constants. When it can, Javelin lets you mix the two, and in general it lets you use a constant in place of a time series. When you cannot mix them, you will get an error message if you try (see Chapter 11, "Special Values and Diagnosing Problems").

In Javelin 1.0 and 1.1, all values are numbers; however, some functions require dates as arguments. (The function SINCE, which tells how long it has been since a given date, is one of these.) Where a function needs a date as an argument, in Javelin 1.0 and 1.1 all you can enter is a date; you cannot enter a variable name since variables cannot contain dates. If you try to enter a variable name, you will get an error (again, see Chapter 11).

However, in Javelin PLUS values can be dates or text as well as numbers. If you give Javelin PLUS one kind of value when it is expecting another, you will get an error. There are functions for changing one data type to another which can be used in these cases. See Chapter 20, "Strings and Dates," for more information.

When you learn to use a particular function, you need to know the name of the function, the number of arguments it accepts, and what type of information each argument must be. If this sounds complicated, don't worry—Javelin helps you type in functions. If you can't remember what a function is called, you can use →|, the Spell key.

After the list of variable names, Javelin adds a list of all its functions; you can use the End key to go to the bottom of this list and work upwards if you don't want to page through all your variable names. Select a function either by typing its name followed by an open parenthesis or by selecting it from the list presented by the Spell key.

Once you have selected a function, Javelin tells you on the information line what kind of argument it expects. As you type, the information line changes to keep you up-to-date on what you should be typing. Also, if you make a mistake (such as too many arguments, not enough arguments, or the wrong kind of argument), Javelin tells you so when you hit ↵ to complete your formula, and it positions the cue at the mistake.

For example, the function SUM accepts a list of arguments. The arguments may be a mixture of constants and time-series data with the same period. A formula containing a SUM function is correct if all its variables are

daily, weekly, monthly, quarterly, or yearly. In the following example, all variables are yearly:

Federal Spending = SUM(Federal Defense Spending, Federal NonDefense Spending)

■ *Results of Functions*

A function is said to *return* an answer: when Javelin calculates the function, it uses the answer where the function was to continue calculating the formula. This means that you can *nest* functions, that is, use a function as an argument to another function. When one function is used "inside" another function, Javelin calculates the inside function first, replaces the inside function with the answer, and goes on to calculate the outer function.

Whether a function returns a time series or a constant depends on the function and the arguments you give it.

■ *Comparing Javelin's Functions to 1-2-3's*

Javelin's functions are very similar to those in Lotus 1-2-3, except the function name does not begin with an "at" sign. There are more functions in Javelin than in 1-2-3, and there are many functions that deal specifically with time periods. Just as 1-2-3 added label functions to its version 2, Javelin PLUS added most of the same text-handling functions.

The Point key allows you point to variables as in 1-2-3, except that it displays names rather than cell addresses.

Types of Functions

■ Javelin's many functions can be divided into the following types:

- Basic functions do simple mathematical operations, such as maximum value, minimum value, and rounding numbers.
- Logical functions do conditional processing based on true/false conditions.

- Time functions convert between variables of different periodicities, and calculate month- or year-to-date sums and other time-related calculations (see Chapter 9, "Javelin and Time").
- Statistical functions do standard deviations and interpolation, and in Javelin PLUS, trigonometric functions and logarithms as well (see Chapter 15, "Statistics").
- Financial functions do net present values, bond prices, and other interest and pricing calculations (see Chapter 16, "Financial Calculations").
- Date-value functions (in Javelin PLUS) handle date values (see Chapter 21, "Javelin Databases").
- Text-value functions (in Javelin PLUS) work with text values, also known as strings (see Chapter 20, "Strings and Dates").
- Cross-tabulation functions (in Javelin PLUS) convert a custom time series to a time series using the dates in a date-value series (see Chapter 21).

See Table 8.7 for a list of functions by type. The Javelin reference manual contains an alphabetical listing of the functions. We'll be discussing functions throughout the rest of this book. Let's start here with basic and logical functions.

BASIC FUNCTIONS (CHAPTER 8)

Function name	Description
ABS	Absolute value.
AVG	Average of two or more expressions.
CHANGE	Change in successive values of an expression.
F	Place-holder for undetermined function.
INT	Integer part of number.
MAX	Maximum of two or more expressions.
MIN	Minimum of two or more expressions.
MOD	Modulo or remainder.

Table 8.7: List of functions by type

PCT	Percentage change in successive values of an expression.
ROUND	Number rounded off to a specified number of decimal places.
SUM	Sum of two or more expressions.

LOGICAL FUNCTIONS (CHAPTER 8)

Function name	Description
AND	Logical *and*.
CASE/ SELECT	Returns one of a list of expressions based on an integer that specifies which selection you want. Changed from SELECT to CASE in Javelin PLUS.
IF	If-then-else.
NOT	Logical *not*.
OR	Logical *or*.

Note: An ***expression*** is a numeric variable or a mathematical combination of variables.

TIME FUNCTIONS (CHAPTER 9)

Function name	Description
A and ACON	Convert an expression to custom period A.
B and BCON	Convert an expression to custom period B.
BEGIN	Returns a portion of a variable's time series, starting at a specified date.
BETWEEN	Number of periods between two dates.
COUNT	Number of periods for which a variable has data.
CUM	Cumulative sum of an expression.
DAILY	Converts an expression to a daily time series.
END	Returns a portion of a variable's time series, ending at a specified date.

Table 8.7: List of functions by type (continued)

GROW	Returns a time series that grows at a specified rate starting at a specified value.
GROWRATE	Compound annual growth rate of an expression.
LENGTH	Number of periods for which a variable is defined.
MDAYS	Number of days in each month.
MONTHLY	Converts an expression to a monthly time series.
MTD	Cumulative month-to-date value of an expression.
NEXT	Value of a time series some periods hence.
PREVIOUS	Value of a time series some periods ago.
QDAYS	Number of days in each quarter.
QTD	Cumulative quarter-to-date value of an expression.
QUARTERLY	Converts an expression to a quarterly time series.
RANGE	Returns the value of an expression for a specified range of time periods.
RAVG	Rolling average.
RMAX	Rolling maximum.
RMIN	Rolling minimum.
RSUM	Rolling sum.
SINCE	Number of periods since a specified date.
TAVG	Average over time.
TMAX	Maximum over time.
TMIN	Minimum over time.
TSUM	Sum over time.
UNTIL	Number of periods until a specified date.
VALUE	Value of expression at a particular time.
WEEKLY	Converts an expression to a weekly time series.
YDAYS	Number of days in each year.
YEARLY	Converts an expression to a yearly time series.
YTD	Cumulative year-to-date value of an expression.
2PREV	Javelin PLUS only; equivalent to Javelin 1.1 PREVIOUS.

Table 8.7: List of functions by type (continued)

2NXT	Javelin PLUS only; equivalent to Javelin 1.1 NEXT.

STATISTICAL AND TRIGONOMETRIC FUNCTIONS (CHAPTER 15)

Function name	Description
ACOS	Arc-cosine (Javelin PLUS).
ASIN	Arc-sine (Javelin PLUS).
ATAN	Arc-tangent, single input (Javelin PLUS).
ATAN2	Arc-tangent, x and y inputs (Javelin PLUS).
CORR	Correlation coefficient for two time series
COS	Cosine (Javelin PLUS).
EXP	Natural antilogarithm.
INTERCEPT	Y-intercept for least-squares line for two time series.
INTERP	Fills in missing values in a time series, based on straight-line interpolation.
LN	Natural logarithm.
NORMAL	Random numbers in a normal distribution (Javelin 1.1).
PREDICT	Predicted value of a time series based on least-squares regression line of that time series and another time series.
RANDOM	Random numbers between 0 and 1 (Javelin 1.1 and later).
SIN	Sine (Javelin PLUS).
SLOPE	Slope of least-squares line for two time series.
SQRT	Square root.
STDEV	Standard deviation.
TAN	Tangent (Javelin PLUS).
TREND	Predicted value of a time series based on least-squares line for that series against time.
TRENDCORR	Correlation coefficient for a time series against time.
TRENDSLOPE	Slope of least-squares line for a time series against time.
TRENDSTART	Y-intercept of least-squares line of a series against time.
VAR	Variance.

Table 8.7: List of functions by type (continued)

FINANCIAL FUNCTIONS (CHAPTER 16)

Function name	Description
BONDPRICE	Price of a bond based on specified yield, face value, coupon rate, years to maturity, and number of payments per year.
BONDYTM	Yield to maturity of a bond.
DEPRDECL	Depreciation for each period using the declining balance method.
DEPRSL	Depreciation for each period using the straight-line method.
DEPRSOYD	Depreciation for each period using the sum-of-the-years'-digits method.
FV	Future value.
IRR	Internal rate of return.
NPV	Net present value.
PMT	Installment payment for the specified principal, rate, and number of periods.
PV	Present value.

DATE FUNCTIONS (CHAPTER 20; Javelin PLUS only)

Function name	Description
BEGINDATE	First date that an expression is defined.
DAY	First day of the date specified by an expression.
ENDDATE	Last date that an expression is defined.
FIRST	First date at which an expression is true.
LAST	Last date at which an expression is true.
MONTH	First month of the date specified by an expression.
QUARTER	First quarter of the date specified by an expression.
TODAY	Current date.
WEEK	First week of the date specified by an expression.
YEAR	First year of the date specified by an expression.

Table 8.7: List of functions by type (continued)

TEXT FUNCTIONS (CHAPTER 20; Javelin PLUS only)

Function name	Description
CHAR	A single character, defined by its numeric (ASCII) code.
CODE	Numeric (ASCII) code of a character.
EXACT	Tests if two strings are exactly the same regardless of capitalization.
FIND	Finds a smaller string in a larger one and returns the position where the smaller string starts.
LEFT	Keeps the leftmost part of a string.
MID	Keeps the middle part of a string.
NUM	Converts a string into a number.
REPEAT	Makes a long string by repeating a shorter string.
REPLACE	Changes a string by replacing part of it with another string.
RIGHT	Keeps the rightmost part of a string.
STRING	Converts a date value or a number to a string.
STRLEN	Length of a string.
TRIM	Drops trailing spaces from a string.
TYPE	Type of expression (1 = number, 2 = string, 3 = date, 4 = constant).

CROSS-TABULATION FUNCTIONS (CHAPTER 21; Javelin PLUS only)

Function name	Description
INDEX	Returns a series of dates in which the values of one series match the values of a second variable.
SELECT	Selects the values of a time-series variable according to a logical selection series.
SORTA	Sorts the values of a variable into ascending order.
SORTD	Sorts the values of a variable into decending order.
XTABAVG	Given a series of dates and a series of values, pairs up dates

Table 8.7: List of functions by type (continued)

	and values, and averages together all values whose dates are the same. Returns a series whose period is that of the dates given.
XTABMAX	Same as XTABAVG, but finds the maximum of the values whose dates are the same.
XTABMIN	Same as XTABAVG, but finds the minimum of the values whose dates are the same.
XTABSUM	Same as XTABAVG, but sums the values whose dates are the same.

Table 8.7: List of functions by type (continued)

■ *Basic Functions*

As noted above, Javelin includes basic functions that accept a single argument, others that accept two arguments, and others that accept an unlimited number of arguments.

Single-Input Functions First, let's look at the basic functions that accept just one argument. These functions can accept either a constant or a time series as input. A constant input produces a constant output and a time-series input produces a time-series output. In this list of functions, *X* stands for the constant or time-series argument that the function will process. Two single-input functions are:

- ABS(X): Returns the absolute value of X. If X is positive, the absolute value is the same number. If X is negative, the absolute value is −X.
- INT(X): Returns the integer part of X, without the decimal part.

As an example, let's create a rounded-off, adjusted GNP variable called Adjusted GNP Int (when Javelin divides one number by another, it keeps up to 15 decimal places of precision, not all of which you may want).

In the border cell above an empty column, type:

Adjusted GNP Int = INT(Adjusted GNP)

(You can use F4, the Point key, when entering this formula.) Calculate the model using F9, the Calc key. Notice that all the values are integers—that is, they have no nonzero digits after the decimal point.

Other single-input functions require a time series as their argument. They are useful for performing simple analysis of time-series data.

- CHANGE(X): For each value of the variable X, calculates the difference between that period's value and the previous period's value.
- PCT(X): For each value of the variable X, calculates the percentage change between that period's value and the previous period's value.

Calculate the change in GNP from one year to the next, both as a dollar amount and as a percentage. Enter these formulas in top border cells:

Change GNP Adjusted = CHANGE(Adjusted GNP)
Pct Change GNP Adjusted = PCT(Adjusted GNP)

Now recalculate the model.

Negative numbers are shown in parentheses. Notice that there is no value for either Change GNP Adjusted or Pct Change GNP Adjusted for 1966. Since these functions compare the value of Adjusted GNP for the current period to the value for the previous period, these functions can return no value for the first value in the series.

Double-Input Functions Other functions need two inputs to do the calculation. Use them when the calculation you want to perform on the first argument depends in some way on the second argument (for example, to round off X to N decimal places). But these multiple-input functions raise three questions about their arguments:

- What happens if one of the arguments is a constant and the other is a time series? If either argument is a time series, the result is a time series; if both arguments are constants, the result is a constant.
- What happens if both arguments are time series but they have different periods (for example, one is weekly and the other is monthly)? Both time series must have the same period. (For information on functions that change the period of a variable, see Chapter 9, "Javelin and Time.")
- What happens if the time series have different lengths? Consult Chapter 9 for a full description of the time span of the result.

Here are two such multiple-input functions:

- MOD(X, M): X modulo M, that is, the remainder that is left over if X is divided by M. M is called the *modulus*.
- ROUND(X, N): X rounded off to N decimal places. If N is zero or missing, X is rounded off to an integer. If N is a positive number, X is rounded off to that many decimal places. If N is a negative number, X is rounded off so that there are N zeros. For example, ROUND(12345,–2) would result in 12300. Even if N is not an integer, only its integer part is used (it's hard to round something to 2½ decimal places).

Above, you rounded off Adjusted GNP to an integer, creating Adjusted GNP in Billions. However, the rest of the data is accurate to one decimal place, so let's use the ROUND function to round off Adjusted GNP to one place.

Enter in a top border cell

Adjusted GNP Rounded = ROUND(Adjusted GNP,1)

Recalculate the model. These numbers look just like Adjusted GNP, but instead of being formatted to display one decimal place, they have no non-zero digits beyond one decimal place.

Change the format of Adjusted GNP Rounded to show two decimal places, using the command / **D**efine **V**ariable. Set Decimal Places to 2 and exit from the settings sheet. The second digit after the decimal point is a zero for every value. If you change the format for Adjusted GNP, you see that it has more significant digits.

Functions to Combine Variables Other functions can be used to combine several constants or several time series. The functions can accept any number of arguments (they require at least one) and, as above, the arguments may be a mixture of time series and constants. The same three issues apply as for "Double-Input Functions" above.

- SUM(X, ...): Sum of the arguments.
- AVG(X, ...): Average of the arguments.

- MAX(X, ...): Maximum value of the arguments.
- MIN(X, ...): Minimum value of the arguments.

You can use the SUM function to combine Federal Defense Spending and Federal NonDefense Spending to make Federal Spending. Let's stop using the Worksheet view and try using a combination of the Formulas and Table views so you can see how creating formulas works using them.

Display two windows on the screen if you only have one displayed, using / **W**indow **2**. In one window, request a Table view of Federal Defense Spending. Switch to the other window and look at a Formulas view of all variables, using the / **V**iew **F**ormulas **A**ll command.

Type in the formula for the new variable, Federal Spending. Use the Spell key or type in the variable names by hand.

Federal Spending = SUM(Federal Defense Spending,
Federal NonDefense Spending)

Note that this is the same as saying Federal Spending = Federal Defense Spending + Federal NonDefense Spending. The SUM function only exists for symmetry with AVG, MIN, and MAX, and for the convenience of 1-2-3 users.

With your cursor still on the formula for the new variable, press F3, the Sync key, to display the same variable in the Table view. Recalculate the model to see the values.

Look at Change GNP Adjusted by moving your cursor in the Formulas view up to it—since the windows are in sync, the Table view displays the values for the same variable. Some years, adjusted GNP shrinks rather than grows.

What if you need a variable equal to the annual change in adjusted GNP, except that when the change dips below zero, you want your variable to be zero? You can calculate this using the MAX function—you want the value of Change GNP Adjusted or 0, whichever is greater. Type

GNP Growth = MAX(Change GNP Adjusted, 0)

Calculate the model with F9, the Calc key. The variable never drops below zero. The same technique can be used with MIN to create a variable with a ceiling value.

■ *Logical Functions*

You may want a variable calculated according to one formula under some circumstances and according to another formula under others. To do these sorts of formulas, use the IF function and *conditions.*

The IF Function IF allows you to say, If a certain condition is true, do this, otherwise, do that. The IF function takes three arguments:

IF(condition, formula if true, formula if false)

The condition, or *conditional expression,* is one that can be true or false. Conditions can contain the following *conditional operators:*

- < Less than
- > Greater than
- = Equal to
- <= Less than or equal to
- >= Greater than or equal to
- <> Not equal to

In Javelin, true is represented by any nonzero number, usually 1, while false is represented by zero. So you can think of these conditional operators as arithmetic operations whose answers are 1 if they are true and 0 if they are false. That's why they are included in Table 8.6 with the other arithmetic operations. For example, 55 > 21 gives 1 (true), and 1976 = 1977 gives 0 (false). (In Javelin PLUS, date and string values are also considered as true.)

If the condition in the IF function is true or 1, the variable is calculated using the second argument, *formula if true,* and otherwise it uses the third argument, *formula if false.*

Let's create a variable called Expected Unemployment that will be 7 percent in case of recession and 3 percent otherwise. The way you will determine whether there is a recession is if the GNP fell—that is, if the change in adjusted GNP was negative.

Enter the following function, using the Point key:

Expected Unemployment = IF(Change GNP Adjusted < 0, .07, .03)

You may want to set the format for this variable to percent by using the Variable settings sheet.

Nesting IFs Conditions can be more complex, requiring two conditions to be true or false. You can set up this kind of formula by nesting IFs as in these examples:

```
IF(condition1, IF(condition2, formula1, formula2), formula3)
IF(condition, formula1, IF(condition2, formula2, formula3))
```

For instance, another way to tell whether there is a recession or a depression is if unemployment is on the rise. Let's create a Recession variable that takes the following values:

- Value = 1: GNP is not shrinking.
- Value = 2: GNP is shrinking but unemployment is steady.
- Value = 3: GNP is shrinking and unemployment is rising by more than 2 percent.

Type in the formula for Recession:

```
Recession = IF(CHANGE GNP Adjusted < 0,
              IF(CHANGE(Unemployment) > .02, 3, 2), 1)
```

Now recalculate the worksheet; only 1975 rates as a recession year.

Complex Conditions You can also use *compound conditions,* using *and, or,* and *not.* These are functions, and the arguments are one or more conditions.

- AND(list of conditions): Returns a true value if *all* the arguments are true; otherwise, returns a value of false.
- OR(list of conditions): Returns a true value if *any* of the arguments are true; otherwise, returns a value of false.
- NOT(condition): Returns a true value if the condition is false, and a false value if the condition is true; changes trues into falses and falses into trues.

Using OR, you can also find out which years were not generally good economically. Calculate which years had a problem, either shrinking GNP or rising unemployment, by using an OR function. Enter the formula for Problem.

```
Problem = OR(Change GNP Adjusted < 0,
              CHANGE(Unemployment) > 0)
```

There are more years that had problems than the ones you identified as recession or depression years.

Calculations Based on Cases Some models require that a variable be calculated a number of different ways depending on the circumstances. You can use the CASE function to choose from a list of formulas or values, based on the value of a *selector.* In Javelin 1.1, this function is called SELECT. Javelin 1.1 models loaded into Javelin PLUS will have this change made automatically. The function works like this:

```
CASE(selector, list of formulas)
```

The selector can be a constant or a time series. If the selector is equal to 1, Javelin uses the first formula from the list. If the selector equals 2, Javelin uses the second formula, and so on. If the selector is less than or equal to zero, or greater than the number of formulas in the list, Javelin uses the last formula in the list. As with ROUND above, Javelin ignores the fractional part of the selector if there is one.

You can base a calculation on the value of Recession, so that the values will be different for normal, recession, and depression years. Try making a variable for predicted growth for the books and records company, in which you are predicting no growth during depression years, 2 percent growth during recession years, and 5 percent growth during normal years.

Enter this formula for Predicted Growth:

```
Predicted Growth = CASE(Recession, 0.05, 0.02, 0)
```

Display this variable as a percentage. Give the command / **D**efine **V**ariable, set the format to percent, and exit the settings sheet.

If you have a long list of formulas or values, if the selector isn't a small number, or if it has a fractional value, you can also use a Lookup Table building block (see Chapter 17, "Building Blocks").

■ The Place-Holder Function

There are times when you know that one variable depends on one or more others, but you do not yet know how. To indicate this, you can use the *place-holder function,* F. By doing this, you tell Javelin to indicate in the Diagram and Formulas views that the dependence exists. Naturally, Javelin can't calculate the variable based on the F function—the values are undefined until you figure out the relationship. This is useful when you are developing a model but don't have it completely worked out.

For example, you may want to calculate a variable Predicted GNP, but you haven't decided how the calculation will work. Since you do know that the inputs will probably be Adjusted GNP and Unemployment, you could set up the following formula. Type

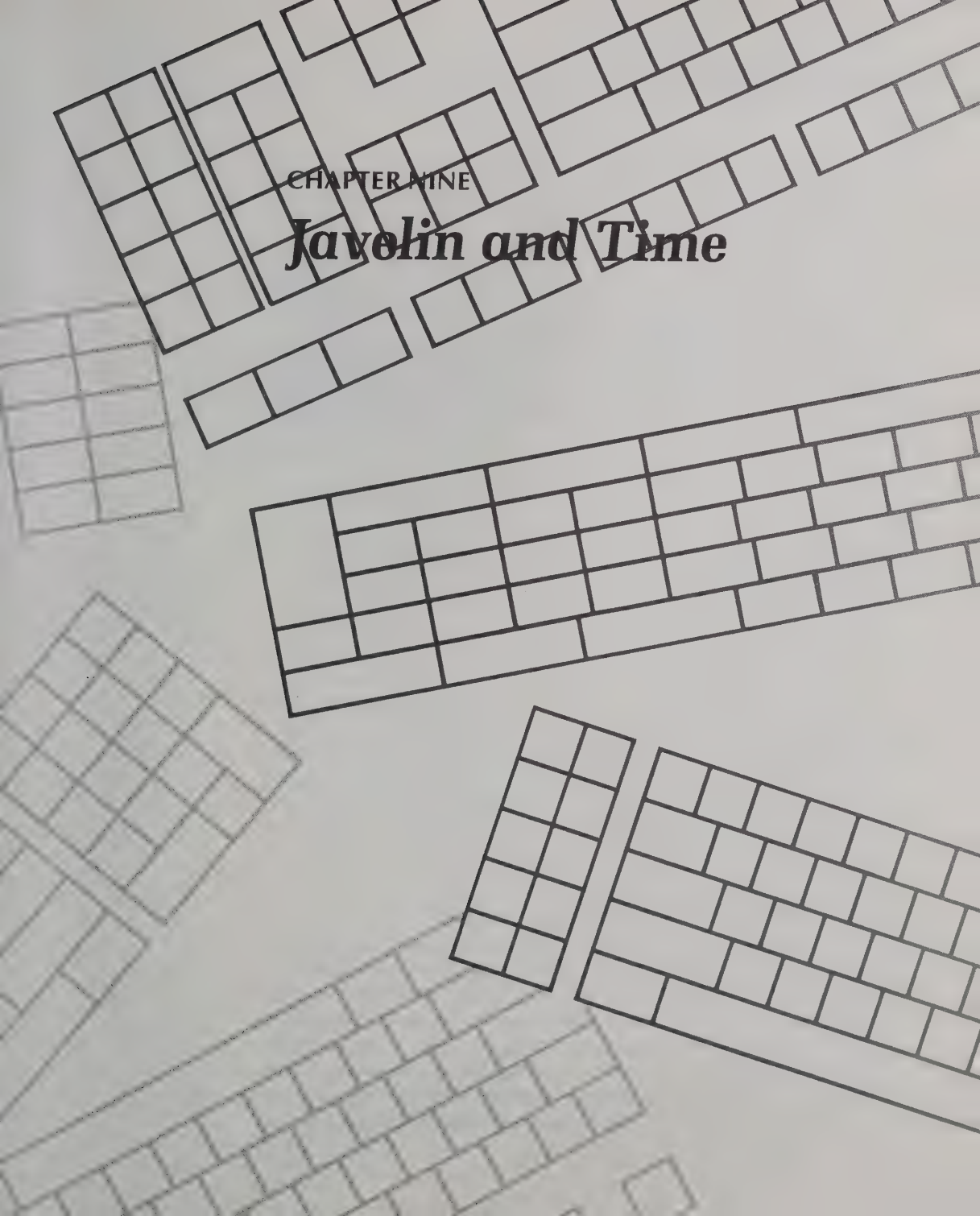
Predicted GNP = F(Adjusted GNP, Unemployment)

Another case in which you will see the F function is if you load a model created in a later version of Javelin into an earlier version. If you create and save a model in Javelin PLUS, using the new functions that exist in PLUS but not in 1.1, when you load that model into Javelin 1.1, the new functions will appear as F. Caution: if you load a 1.1 or PLUS model that includes new functions into Javelin 1.0, you will get an error message, Javelin may "freeze," and you will probably have to turn your computer off.

Save this model again as ECON, because you will use it in later chapters.

Conclusion

■ You have seen how to use Javelin's basic, logical, date value, and text value functions. There are also a number of functions you can use to calculate formulas that have to do with time: month-to-date sums or number of periods in a range of time. In the next chapter, we will cover these time functions and show how Javelin works with time.



CHAPTER NINE

Javelin and Time

One thing that distinguishes Javelin from other programs is its built-in calendar. You have used Javelin's calendar in the models made in previous chapters; now it is time to look more closely at how it works. In this chapter we will talk about dates (typing them in and which ones Javelin knows about), time spans (what range of time your model and its variables cover), and time conversions (how Javelin groups smaller periods into larger periods and how it splits larger ones into smaller ones).

We will also introduce Javelin's time-handling functions. While Javelin does much work with time automatically, sometimes (in a formula) you must tell it explicitly what to do; that is what the time-handling functions are for. As we go along, you will build a model of a chain of fast-food restaurants that we'll call Mr. Pizza. This model uses many of the time features of Javelin and will be used again in Chapter 17, "Building Blocks."

The Javelin Calendar

■ Javelin has a calendar that runs from January 1, 1904, to December 31, 2082, long enough for most applications. Within this calendar, Javelin divides time into six different periods: days, weeks, months, quarters, years, and constant (one value carried across the whole calendar), as well as two *custom periods*. Any date you type in or Javelin displays is expressed in terms of a time period. For example, 7 December 1941 is a daily date; 12 Week 1954 is a weekly date; Mar 1958 is a monthly date; 3 Quarter 1962 is a quarterly date; and 1995 is a yearly date.

■ Typing in Dates

When you type in a date, you can abbreviate considerably: Week and Quarter can be abbreviated to W and Q, and month names can be abbreviated to three letters. For convenience, the years 1950 to 2049 can be abbreviated to one or two digits, so the date 87 means 1987 and 3 means 2003.

This can be a trap for the unwary. For instance, when typing entries into a worksheet border, if you type 0, Javelin interprets it to mean the year 2000. These year abbreviations are handy, so use them carefully—check to see that you get the year you intend. This is easy to do because, no matter how you

abbreviate the date, once Javelin recognizes it, it replaces it with the standard display for that date.

■ Yearly Dates

Javelin years run from January through December, but Javelin can also handle fiscal years that start and end at other times (described below). Regardless of when your model's fiscal year ends, a month such as Oct 1987 is always the same calendar month. However, if you change your fiscal year end, Oct 1987 could be part of fiscal 1988 or 1986.

■ Quarterly Dates

A quarter is simply three months. The first quarter of a year is January through March, the second quarter April through June, the third quarter July through September, and the fourth quarter October through December, unless you change the fiscal year end of a model. If you change the fiscal year, the first quarter is the first three months of the fiscal year.

■ Weekly Dates

Javelin weeks run from Sunday to Saturday and are numbered from the first week in the year to the 52nd or 53rd, depending on the year. The first week in a year is the week ending on the first Saturday in the year. For example, since 1 Jan 1983 fell on a Saturday, the first week in 1983 is considered to be the week from 26 Dec 1982 to 1 Jan 1983, the second week in 1983 is the week from 2 Jan 1983 to 8 Jan 1983, and so forth to the 53rd week of 1983, which runs from 25 Dec 1983 to 31 Dec 1983.

This scheme for numbering weeks can lead to some peculiar situations. For example, all workdays in 1 W 83 (26 Dec 82 to 1 Jan 83) are actually in 1982. On the other hand, this scheme is as easy to understand as any. Javelin has a setting called the *splitting method* (described below), which lets you deal with situations in which it is important that a week be considered partly in one year or month and partly in another year or month.

The rules for converting between weeks and larger periods are complicated since there is usually not an even number of weeks in a month, quarter, or year. We discuss them below under "Time Conversion."

In Javelin PLUS, you can have weeks end on a day other than Saturday. On the Model settings sheet, there is a Week Ends setting that you can set to any day of the week.

■ *Dates with Custom Periods*

If none of the built-in time periods meets your needs in a model, you can create your own using Javelin's custom periods. These can be some subdivision of regular periods (like two halves in a year), or they can have no relationship to time (like check numbers in a check register). If your custom period is related to time, like the half year, you might give it a name like Half; Javelin would display 1 Half 1986, 2 Half 1986, and 1 Half 1987. In the second case, if you named the period CheckNum, a "date" (really an instance of CheckNum) would appear as 105 CheckNum. Custom period dates behave in most ways like regular dates, so most of the material in this chapter applies equally to them and to regular dates. We discuss them in more detail in Chapter 10, "Custom Time Periods."

The Sample Model: Mr. Pizza

■ The Mr. Pizza model is a cash flow model. It begins as a simple model of one pizza joint and becomes more sophisticated as you build up a chain of pizzerias. You'll start with the income and expense variables and calculate the variable Cash On Hand, which is last month's cash plus this month's income minus this month's expenses. Later on, you'll improve the formulas so that income includes interest from the bank (on deposits) and expenses include both operating expenses and startup expenses for other restaurants.

You will start the model as we talk about time spans; once you get to time conversions, you can look at how this affects your results. Finally, with time functions, you can finish the model.

Time Span of the Model

■ This is probably the single easiest place to trip up when creating a model in Javelin: every Javelin model has a *default time span*. It is important to know

that *your model will probably give wrong answers if you have any values outside this default time span.* If you have data for 1985 (and formulas that refer to those data), but your default time span begins in 1986, you may not get any answers for 1985. Although Javelin uses a simple rule to decide whether it can calculate a value at a particular date, as soon as you apply that rule to several formulas (as in a model), things begin to get complicated. The simple rule is as follows: If all data needed to calculate an answer exist for a given date, Javelin will calculate an answer for that date. Thus, a formula like

```
MPG = Miles Driven / Gallons Used
```

will calculate an answer at all dates for which there are both Miles Driven and Gallons Used data.

If not all the data needed to calculate the answer at a date exist, the default time span becomes important. If that date is *within* the default time span, Javelin will convert constants to time series, use functions that create data, and in general do what you specify in your formula. For example,

```
MPG = (Miles Driven / Gallons Used); 25
```

will give the answer 25 for MPG for dates within the default time span where either Miles Driven or Gallons Used does not exist (the ; operator is described later in this chapter). Another example,

```
Sales = Grow(1000 @ 1 January 1987, .01)
```

gives a series that starts at 1000 on 1 January 1987 and grows by 1 percent a day (the GROW function is also described in this chapter). But that series stops at the end of the default time span.

Neither of these two examples will give an answer if the date being calculated is outside the default time span. We've never met anyone who could reliably work out the implications of these rules for a given model. Therefore, you should make the default time span longer (starting earlier, ending later) than absolutely necessary. This may make your model slower to recalculate, but it is safer. Once you have good results from your model, look at the time spans of your data and formulas (/ ***F**ormulas **T**ime spans in the Formulas view). You can then change the default time span using / **D**efine **S**ettings. Otherwise your model may work well enough to make you think you are getting proper results but not, in fact, well enough to give you the right answers.

Having said all this, we should add that several subtle changes were made to Javelin PLUS, which combined to make it more forgiving about calculation outside the default time span. It is, however, still advisable to set the time span correctly on the Model settings sheet.

When Javelin starts up with an empty model, the default time span is set to twelve months, starting at the current month (as determined by the computer's internal clock) and ending 11 months hence. This is almost certainly not the right time span except for quick "back of the envelope" models. You will almost always want to use the Model settings sheet (/ **D**efine **S**ettings) to change the model's default period and time span.

Javelin also has a *default period* (months), which is used only as a suggested period when you create a new variable. The right default period for your model depends on the kind of model you are building. If the model works mostly in terms of years, set the default period to years. If the model works mostly in weeks, set it to weeks, and so forth. If the model doesn't relate to time very often, set the default period to constant or perhaps to a custom time period if you are using one in your model.

Starting the Mr. Pizza model illustrates this. If you have been doing other work in Javelin, save and clear your model. Use the command / **D**efine **S**ettings to look at the Model settings sheet.

Change the model name to MRPIZZA. The default period should be months, the starting month of the model should be Jan 88, and the ending month should be Dec 88. Take this opportunity to make the default format Dollars as well.

Time Spans of Variables

■ Every Javelin variable has a time span—either the variable is constant or it is a time series with starting and ending dates. If the variable is constant, it has the same value for all dates. A typical constant might be an interest rate that is the same every month or the number of tons of coal needed to make a ton of steel. If it's not a constant, then a variable is a time series, representing values at different dates and containing many values.

■ *Input Variables*

An input variable is a variable into which you type data, so it's easy to tell its period and time span: the period corresponds to the period of the data typed in, and the time span starts at the time of the first value typed in and ends at the last. For example, the variable Income does not yet exist in your Mr. Pizza model. Let's define it, first giving the command / **V**iew **T**able and typing Income.

Javelin shows you a monthly table, starting at the current month. Let's enter data for periods that are outside the default time span so you can see what happens. When you enter the value for January 1986, Javelin makes Income monthly and makes its time span one month, with both the start and end dates being January 1986. If you then enter a value for March 1986, Income's end date becomes March 1986. Javelin automatically makes the time span of an input variable large enough to include whatever values you have entered. Type sample data into Income so that you have the following values:

Period	You type
Jan 1986	**10000**
Feb 1986	**11000**
Mar 1986	**12000**
Apr 1986	**13000**
May 1986	**14000**
Jun 1986	**15000**
Jul 1986	**16000**
Aug 1986	**17000**
Sep 1986	**18000**
Oct 1986	**19000**
Nov 1986	**20000**
Dec 1986	**21000**

Why did Javelin pick a monthly period for Income? Because the default period for the model is monthly. When you view a new variable in the Table, Chart, or Quick-graph view, Javelin guesses that you want its period to be the model's default period. Let's define a new variable, Expenses, with the command / **V**iew **T**able Expenses.

As with Income, Expenses comes up as monthly. But let's say that you have only a yearly estimate for expenses. *Before you enter any data* for

Expenses, change its period to Year. If you attempt to change the period of an input variable after you have entered data for it, Javelin will do so, but only if you allow it to delete all the data you've entered for that variable! This is rarely what you want to do. To change Expenses to yearly, give the command / **D**efine **V**ariable Expenses, select Year as the default period on the Variable settings sheet, and exit the settings sheet. Now the Table view lists years rather than months.

There is another, perhaps simpler way of making Expenses a yearly input variable. Let's redefine Expenses that way. First, clear the definition you have, using the command / **C**lear **V**ariable Expenses. Now, switch windows (press F6) to leave the Table view on the screen, give the command for a Worksheet view of Mr. Pizza: / **V**iew **W**orksheet, and press ⟵.

When you enter data on a worksheet, the period of a variable is set to agree with the date for which you first enter a value. For example, type

```
Expenses
```

in the top border of the first column. In the left border of the first row, enter

```
1986
```

Now enter

```
120000
```

in the body cell, and Expenses is made a yearly variable, with your entry its value for 1986. Use F3, the Sync key, so that the Table view in the other window shows the view for Expenses. You can then enter other values for Expenses in a worksheet or any other view.

The easiest way to define a constant is by typing it as a formula. This both sets the value and makes the variable a constant. Display a Formulas view using / **V**iew **F**ormulas **A**ll; let's set the interest rate for funds on deposit at 8 percent, which you'll use later to calculate the interest paid by the bank. Type

```
Interest Rate = .08
```

■ *Calculated Variables*

The period and time span of a calculated variable are determined from the variables and functions that define them. A description of the rules for

determining the time span of a calculated variable looks somewhat complicated, but generally Javelin does what you would expect: when variables are added, subtracted, compared, or otherwise combined, the result's time span is the intersection of the time spans of the input variables. That is, the start date of the result is the latest of the start dates of the inputs and the end date of the result is the earliest of the end dates of the inputs.

You can illustrate this by including income from a proposed second restaurant in your model. Let's say that you expect it to open in October; call the variable Income2. To create the Income2 variable, give the command / **V**iew **T**able Income2 and enter the following values:

Period	You type
Oct 1986	**500**
Nov 1986	**600**
Dec 1986	**700**
Jan 1987	**800**
Feb 1987	**900**
Mar 1987	**1000**

You can now define Total Income by entering its formula and calculating the model. Type

Total Income = Income + Income2

Recalculate with F9. Though there are data for Income from January through December 1986, and Income2 has data from October through March 1987, Javelin 1.1 gives results for only October, November, and December—the months in which there are data for *both* Income and Income2. Javelin 1.1 computes results only when it has all the inputs; that's why Total Sales starts in October and ends in December. Javelin PLUS gives results for the entire time span January 1986 to March 1987.

The ; Operator

■ If you want Javelin to make an assumption about periods outside the time span of a variable, use the semicolon operator. For each date at which Javelin calculates the formula, it will use the value to the left of the semicolon, if one exists at that date; otherwise it uses the value to the right.

(The exact definition is that the value at each date is the expression to the left unless its value is #BLANK (#RANGE in Javelin 1.1), in which case it's the expression to the right. See Chapter 11, "Special Values and Diagnosing Problems," for a definition of #RANGE.)

If you want Javelin to assume that Income2 is 0 in months for which you don't have data, redefine Total Income as

Total Income = Income + Income2; 0

Then recalculate with F9. But no additional results appear for Total Income. This is because the time period for the model is Jan-Dec 1988, while Income and Income2 have data from 1986. Javelin uses the value after the semicolon only within the default time span. Because your data are in 1986 and your default time span is in 1988, *; 0* added to Income2 doesn't solve your problem. This is the single most common error when using Javelin, and that's why we illustrate it here. This is also why Javelin PLUS handles it differently. In Javelin PLUS, addition and subtraction treat missing values (#BLANKs) as zero, while multiplication and all other operations involving #BLANKs give #BLANK as their result. In Javelin PLUS it is still a good idea to use the semicolon to tell Javelin exactly what you mean.

Let's fix the model's time span on the Model settings sheet so that it includes the periods for which there are data. Give the command / **D**efine **S**ettings; set the beginning date to October 1985 and the ending date to December 1987. Then recalculate (F9) and use the Home key to see all the results for Total Income in the Table view.

You can also use the semicolon to handle actual and estimated values. For example, in a formula like

Sales = Actual Sales; Estimated Sales

the value of Sales is the actual value if there is one, otherwise it's the estimated value.

The time span of the Total Income variable is now determined by the time span of Income, because whenever there is no value for Income2, Javelin uses 0. Some functions let you choose explicitly the time span of a variable to use—BEGIN, END, and RANGE. Other functions (like PREVIOUS and NEXT) take input data and shift each value forward or backward in time; they therefore shift the time span of their input variable forward or backward. We will go into more detail on time spans as we discuss functions and operations that affect them.

Time-Handling Functions

- Javelin has several kinds of functions that have to do with time:
  - Time-shifting functions: Move the values of a time series forward or backward some number of periods.
  - Time-conversion functions: Convert the value of a time series to another periodicity, including combining small periods into larger ones, or splitting large periods into smaller ones.
  - Accumulation functions: Report cumulative sums, counts, and other totals for a time series. These include period-to-date functions that result in a time series, as well as functions that result in a constant.
  - Extrapolation and interpolation functions: Invent data to fill in the gaps of input variables.
  - Day-count functions: Count the numbers of days in larger periods and are useful for date arithmetic.
  - Time-span limiting functions: Limit the time span of a time series.
  - Smoothing functions: Calculate the rolling sum, average, maximum, or minimum for a time series. These combine the values of time series from several different periods.
  - Rate functions: Calculate the relative or absolute rate of change of a time series.
  - Date arithmetic functions: Count periods since, until, and between dates.
  - Date functions: There are also new time-related date functions in Javelin PLUS (see Chapter 20).

Let's look at each of these groups of functions.

Time-Shifting Functions

- The two time-shifting functions PREVIOUS and NEXT allow you to create models where the result in one period depends on the situation in

another period. For example, the amount of interest paid on a bank balance this month depends on what the balance was last month, so you use PREVIOUS to get the previous month's balance. Another place you might use NEXT and PREVIOUS is in a model of the number of people employed in a business: you might need to order office furniture two months in advance of hiring people, to allow for delays in shipping. In that case you would use NEXT to get the number of people to be hired two months hence.

Here is how the two functions work:

- PREVIOUS(expression, N periods back): For each period, PREVIOUS returns the value that the expression had N periods ago. For example, in September the result of PREVIOUS(Income,1) is the value that Income had in August. *N periods back* is optional; Javelin assumes one period unless you tell it otherwise.
- NEXT(expression, N periods forward): For each period, NEXT returns the value that the expression will have N periods from now. In September, NEXT(Income,1) gives the value that Income will have in October. As with PREVIOUS, *N periods forward* is optional; Javelin assumes one period.

The expression you use should be a time series, since time-shifting a constant accomplishes nothing. The optional number of periods to shift usually should be a constant. If the number of periods to shift is specified as a number or a constant-valued variable, or is omitted, the time span of the result of the function is the time span of the input shifted appropriately. For example, if the time span of the input were March through December and the number of periods to shift were 2, the time span of the result of PREVIOUS would be January through the following October, or the time span of NEXT would be May through February.

Except in Javelin 1.0, the number of periods to shift can be a time series rather than a constant, but this is very tricky. In particular, the value of the function at January depends on the value of the N series at January, not at the month used to get the value. Also, if N is a series, the time span of the result of PREVIOUS or NEXT is the same as the time span of N. This often causes problems at the end of the time span, as the last few points in the result series get lost. If this sounds confusing, that's because it is. We have seen very few models that really need variable time-shifts.

You will use the PREVIOUS function several times in the Mr. Pizza model before you are through. For starters, define the interest the bank paid you for the money you had on deposit last month (which will be stored in the variable Cash). Since you're in the Table view, just type the formula:

```
Interest = PREVIOUS(Cash; 0, 1) * Interest Rate / 12
```

Divide the interest rate by 12, since it is an annual rate and Cash is a monthly variable. Otherwise you end up applying an 8-percent interest rate 12 times a year and make 96 percent per year!

Cash is last month's Cash (if any) plus this month's Cash Flow. To define it, type

```
Cash = PREVIOUS(Cash; 0) + Cash Flow
```

Remember that using PREVIOUS(Cash) and PREVIOUS(Cash, 1) is equivalent. Cash Flow you will define as Total Income less Expenses. However, you defined Expenses as a yearly variable, while Total Income is monthly. That means you must think about converting the yearly Expenses into monthly.

The PREVIOUS and NEXT functions work slightly differently in Javelin PLUS than they did in Javelin 1.1. Many Javelin 1.1 models contain formulas like this:

```
Balance = PREVIOUS(Balance) + Deposits ; 1000
```

In Javelin 1.1, for the first period PREVIOUS( ) returns a #RANGE. The result of adding #RANGE to anything is #RANGE, so for that period the value of Balance will be 1000. In Javelin PLUS, the first value of PREVIOUS( ) is #BLANK, which when added to Deposits gives the value of Deposits, and will not ever look at the 1000.

To allow Javelin 1.1 models to continue to work in Javelin PLUS, Javelin PLUS adds two new functions ZPREV( ) and ZNXT( ), which behave like Javelin 1.1 PREVIOUS( ) and NEXT( ). If you load Javelin 1.1 model into Javelin PLUS, the PREVIOUS and NEXT( ) functions are automatically converted into ZPREV( ) and ZNXT( ).

Try to write your models so that they will work equally well in either version. In this case, a better formula would be:

```
Balance = PREVIOUS(balance; 1000) + Deposits
```

Time Conversion

■ There are two ways to make Javelin do time conversions: try to use variables with different periods in the same formula, or show data on a worksheet in some period other than the one for which it is defined (showing annual data monthly, for instance). Let's look at both kinds of conversions.

■ Automatic Time Conversion on a Worksheet

First, display the worksheet you used earlier. Although you entered Expenses as an annual figure, you can report on it monthly. Move to an empty row in the left border and insert headings for January through June 1986. Values appear for Expenses, even though it is a yearly variable. Javelin has automatically converted it to a monthly value *for display purposes only.*

■ Time-Conversion Functions

You can make Javelin do the same conversion in a formula. Since Cash Flow is income minus expenses, let's define this formula in the Worksheet view. Move to the top border, to the column next to Expenses, and type

Cash Flow = Total Income – Expenses + Interest

Recalculate with F9. These values come up as #UNDEF (undefined) because Total Income is monthly and Expenses is yearly. Javelin cannot do an implicit conversion (as it did for Expenses on the worksheet) because it doesn't know if you want Total Income to be converted to yearly or Expenses converted to monthly; you have to tell it. Modify the formula for Cash Flow so that it converts Expenses to monthly values. Move down to a data cell and edit the formula to read

Total Income – MONTHLY(Expenses) + Interest

Again, recalculate with F9. Values appear for Cash Flow. The values used to calculate Cash Flow are the same values displayed for monthly expenses.

Here are the functions you can use to request a period conversion in a formula:

- DAILY(expression): Converts to a daily series.
- WEEKLY(expression): Converts to a weekly series.
- MONTHLY(expression): Converts to a monthly series.
- QUARTERLY(expression): Converts to a quarterly series.
- YEARLY(expression): Converts to a yearly series.
- A(expression) and B(expression): Converts to an A or B series (A and B are custom time periods).

These time-conversion functions allow you to refine your definition of monthly Interest to account for compounding the interest daily. Right now,

Interest = PREVIOUS(Cash; 0, 1) * Interest Rate / 12

Javelin knows that Interest is monthly because Cash is monthly; Interest Rate is a constant, so it does not affect the time period of Interest. What would happen if you changed Cash to daily? You would want Interest to become daily as well, and it would. However, your formula would give the wrong answer, because Interest Rate / 12 is a monthly interest rate, not a daily one. So if you changed Cash to daily, you'd also have to rewrite the Interest formula. Forcing yourself to remember to do this is bad practice.

Fortunately, Javelin can help here. Let's change the Interest formula. Switch to the Formulas view and enter this new formula:

Interest = PREVIOUS(Cash; 0, 1) * MONTHLY(YEARLY(Interest Rate))

YEARLY(Interest Rate) tells Javelin that this is a yearly rate; MONTHLY (YEARLY(Interest Rate)) tells Javelin to convert that yearly rate to a monthly rate (the same as dividing by twelve). However, if you change Cash to daily, this formula cannot be calculated because MONTHLY(YEARLY(Interest Rate)) results in a monthly value. #UNDEF is your reminder to revise the formula—in this case, to DAILY(YEARLY(Interest Rate)).

By the way, in this example Javelin has to make an arbitrary decision; what is the time span of MONTHLY(YEARLY(Interest Rate))? It is the default time span of the model.

So, there are two ways of asking Javelin to do a time conversion: explicitly (as you just did in the Interest formula) and implicitly in a worksheet, for display only. But no matter how you ask for the time conversion, Javelin always converts the data for any given variable the same way. You tell Javelin what method to use on the Variable and Model settings sheets.

Before continuing with Mr. Pizza, let's digress to review splitting and combining time periods in full detail; they are at the heart of Javelin's power as a modeling system.

■ *Splitting Larger Time Periods*

In converting larger time periods to smaller ones, there are three possible conversion methods. The standard method, *Divide,* breaks down the larger period's value proportionately for the smaller periods. This is what you did for Expenses: each month's value is 1/12 of the annual value for the year. That's why each month came out to be $10,000. However, if the Split by Days setting is set on the variable's Variable settings sheet, Javelin divides up the variable depending on the number of days in the larger and smaller periods. Let's illustrate this for Expenses.

Use / **D**efine **V**ariable Expenses to look at the definition for Expenses. Toggle the splitting method from Split Evenly to Split by Days and exit from the settings sheet. Now, the value for MONTHLY(Expenses) for April 1986 is 30/365 of the annual value, because 1986 has 365 days and April 1986 has 30 days.

Split by Days in this case is fairly exotic. Most likely, you will find it useful for values like Electricity Used where the value is actually dependent on the number of days in the periods involved, and you would expect to use more in a 31-day month than in a month of 30 or 28 days. Change Expenses' splitting method back to Split Evenly.

Table 9.1 shows the effect of different splitting methods. In the first two data columns, you can see the result of splitting yearly data into monthly by dividing, both dividing evenly and splitting by days. In this worksheet, all variables are yearly, so the monthly values you see are computed by conversion. (We chose the value 7300 for the Split by Days variables so that the Split by Days conversion comes out with integer results.)

The second method, *Interpolate,* adjusts larger period values to create the smaller period values. This "smooths" the data to some degree. Say you have an annual series Tons of Coal on Hand and convert it to monthly by interpolating. Then for January 1987, the monthly value is 1/12 more than

the base yearly value for 1987, the February value adds another 1/12, and so forth, until the value for December reaches the yearly value. If the variable is set to Split by Days, rather than adding 1/12 of the base value each month it takes into account the number of days in the month, so that the January 1987 value would be 31/365 more than the base 1987 value, the February value 28/365 more than the January value, and so forth. The third and fourth data columns of Table 9.1 show the effects of converting by interpolating evenly and by days. The values for 1986 are 0.

The last splitting method, *Repeat,* repeats the data for each smaller period in the larger period. If you have a quarterly variable Tunafish Price per Can and you want a monthly series, splitting by Repeat would give the same price for each month in the quarter. The last column in Table 9.1 shows the results of splitting yearly data to monthly using the Repeat method.

The choice between splitting by interpolating and by repeating is frequently an arbitrary one, depending on whether you think of your data values as moving smoothly from one period to another.

	DIVIDE		INTERPOLATE		REPEAT
	Evenly	**by Days**	**Evenly**	**by Days**	
1987	7200	7300	7200	7300	7000
Jan 1987	600	620	600	620	7000
Feb 1987	600	560	1200	1180	7000
Mar 1987	600	620	1800	1800	7000
Apr 1987	600	600	2400	2400	7000
May 1987	600	620	3000	3020	7000
Jun 1987	600	600	3600	3620	7000
Jul 1987	600	620	4200	4240	7000
Aug 1987	600	620	4800	4860	7000
Sep 1987	600	600	5400	5460	7000
Oct 1987	600	620	6000	6080	7000
Nov 1987	600	600	6600	6680	7000
Dec 1987	600	620	7200	7300	7000

Table 9.1: Examples of splitting time periods

■ Combining Smaller Time Periods

The standard method to convert smaller periods to larger ones is to add them up. If you enter

Annual Income = YEARLY(Total Income)

Javelin adds up the twelve monthly Total Income values for each year to compute the Annual Income. This conversion method is called the *Sum* method and is appropriate for variables that represent flows of money, material, or other things, rather than stocks.

If you have a variable like Cash that represents the monthly bank balance for Mr. Pizza and you want to convert that to a yearly value, it would not be meaningful to add up the monthly values. Javelin provides two combining methods to deal with this kind of data.

The *Average* method takes the average of the smaller period's values for the larger period's value. In this case, it would take the average of the monthly balances to be the year's value, which would be useful to model a situation that, for example, requires minimum average balances to be maintained for the year.

The *Last Period* method uses the last smaller period's value to be the larger period's value. In this case, the monthly balance at the end of the year (i.e., December) would also be the year's value. This would be useful in a report involving closing balances for the year.

Table 9.2 shows three monthly variables and the result of converting each of them to yearly, using the three different combining methods. The worksheet shows three monthly variables named Sum, Average, and Last Period, which are set to use the corresponding conversion methods. In the first column it adds up the 12 values for January through December 1987 to get the 1987 value; in the second column it takes the average of the monthly values; and in the third column it uses the value from December.

■ Combining Weeks into Larger Periods

Combining weeks into larger periods has special problems, since weeks do not fit neatly into months, quarters, or years. By default, when summing or averaging weeks into larger periods, the value for each week is assigned to the month, quarter, or year in which the week ends. You can also set the Split by Days option either for a variable or for the whole model, which

makes it assign values proportionally to the number of days of the week that fall in each larger period. For example, 1 W 85, the first week of 1985, actually runs from 30 Dec 1984 to 5 Jan 1985, so two of its days are in 1984 and five are in 1985. If you convert by summing a weekly variable to monthly and set Split by Days, 2/7 of the value for that week is added into Dec 1984 and 5/7 into Jan 1985. If you do not set Split by Days, the entire value for that week is added into Jan 1985, since that is the month in which the week ends.

This complicated example should warn you to be very careful when converting between weeks and larger periods—your answers may not be what you expect. Table 9.3 shows the result of combining weekly data into monthly data, using various combining methods. The worksheet shows five weekly variables, again with names corresponding to their combining methods. In 1987, there are five weeks that end in Jan, called 1 W 87 through 5 W 87. In the first column the data are summed splitting evenly, so Javelin adds together the value of the five weekly data points to get 35.

	Sum	Average	Last Period
Jan 1987	10	10	10
Feb 1987	20	20	20
Mar 1987	30	30	30
Apr 1987	40	40	40
May 1987	50	50	50
Jun 1987	60	60	60
Jul 1987	70	70	70
Aug 1987	80	80	80
Sep 1987	90	90	90
Oct 1987	100	100	100
Nov 1987	110	110	110
Dec 1987	120	120	120
1987	780	65	120

Table 9.2: Examples of combining time periods

The second column uses Split by Days, taking into account the fact that the first four days of 1 W 87 were actually in December 1986, so it computes a value of 4 for Dec 86 and 31 for Jan 1987. The next two columns show the corresponding values for Average, and the last column shows the value for Last Period conversion.

Here is the technical explanation of how Javelin computes the peculiar-looking values in the Average columns. In the Average Evenly column, it sums up the five weekly values for Jan 87 and divides by five, so (2+4+6+8+10)/5 is 6, and that is the answer. For Dec 1986, the only weekly value is 0, so that is the answer.

In the by Days column, Javelin takes into account the fact that only 3/7 of the first week is actually in Jan 1987, and that Jan is only 4 3/7 weeks long. So, it computes ((2*3/7)+4+6+8+10)/(4+3/7) and gets the answer 6.5161290323. The value for Dec is problematical. In Javelin 1.0 and early copies of Javelin 1.1, the values computed when converting weeks to larger periods are wrong when Split by Days is set and there are not weekly data for the full larger period. Evidently nobody ever noticed this problem until one of the authors of this book discovered it and pointed it out, which suggests that Split by Days averaging is not used often.

What this really means is that, again, you should be careful if your model depends on the subtleties of conversions between weeks and larger periods. You might want to use a 5-4-4 calendar, described in Chapter 10, "Custom Time Periods," to make your weeks fit evenly into months.

Weeks	Sum		Average		Last
	Evenly	by Days	Evenly	by Days	Period
52 W 1986	0	0	0	0	0
1 W 1987	7	7	2	2	2
2 W 1987	7	7	4	4	4
3 W 1987	7	7	6	6	6
4 W 1987	7	7	8	8	8
5 W 1987	7	7	10	10	10
Dec 1986	0	4	0	.4705882	2
Jan 1987	35	31	6	6.516129	10

Table 9.3: Examples of combining weekly data

Fiscal Years

■ Normally, when Javelin converts monthly data to yearly data, it takes the twelve calendar months (January through December) and applies the combining method specified by the variables involved. But you may use a *fiscal year,* still twelve months long, but running, for example, from October through September rather than from January through December; in fact, October through September is the United States Government's fiscal year. So to sum up the U.S. Government budget for 1986, you would need to sum up October 1985 through September 1986. Javelin lets you define the fiscal year by specifying its last month.

You define a fiscal year on the Model settings sheet (/ **D**efine **S**ettings). In the lower-left corner of the settings sheet is an entry for the end of the fiscal year, as you see in Figure 9.1. The fiscal year must end at the end of a month, and that month must be within 11 months of the calendar end of the year (you can't have fiscal 1988 ending in 1990). For the U.S. Government fiscal year, you would set the end of FY1988 (fiscal year 1988) to end in September 1988. (Javelin shows the day that the year ends, but it's quite sufficient to type in the month or use the Spell key to see the possibilities.)

The fiscal year end setting affects the definitions of years and of quarters. If you set FY1988 to end in September 1988, then 1Q88 would be October 1987 through December 1987, 2Q88 would be January 1988 through

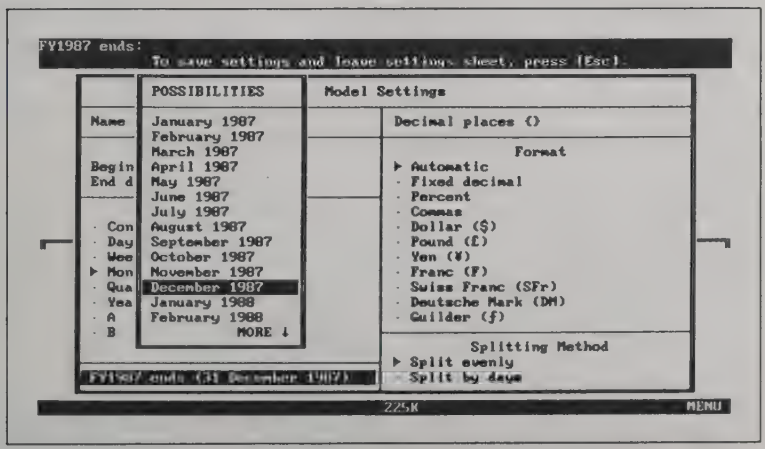


Figure 9.1: Fiscal year end setting with possibilities menu

March 1988, 3Q88 would be April 1988 through June 1988, and 4Q88 would be July 1988 through September 1988. These are called *fiscal quarters.* The year 1988 would be October 1987 to September 1988. The months still represent themselves—that is, December 1987 is still December 1987 even though it happens to be in fiscal 1988. Similarly, 1W88 is still the first week that ends in January 1988. This is standard practice and avoids driving you bonkers trying to keep track of dates.

■ *Continuing the Mr. Pizza Model*

Let's review the Mr. Pizza model you have built. Mr. Pizza started up in January 1986; you want to model monthly cash flow for the year 1986. Income starts at $10,000 per month and increases by about $1000 per month, to $21,700 from two stores by December 1986. Expenditures are $120,000 per year, divided evenly by month ($10,000 per month). The bank pays interest on balances at 8 percent a year. This model shows what your monthly cash flow is and what your monthly bank balance is.

Figure 9.2 shows a Roots view with the equations in the model, and Table 9.4 shows a worksheet with the monthly results. In this model, you have set the model's default range to October 1985 through December 1987, and the default period is months. The variable Cash shows the cash on hand each month. Since Cash is last month's Cash plus this month's Cash Flow, you use the semicolon operator to tell Javelin to use the value 0 for Cash at the beginning of the series. This means that PREVIOUS(Cash; 0) has a value all across the model's default time span: it's either the previous value of Cash or 0. The time span of Cash is therefore determined by the time span of Cash Flow, which runs from January to December 1986.

This model is not very elegant, nor very realistic. However, several more Javelin functions can improve it greatly.

```
Cash = PREVIOUS(Cash;0) + Cash Flow
. Cash Flow = Total Income - MONTHLY(Expenses) + Interest
. . Total Income = Income + (Income2;0)
. . . Income = Monthly data: January 1986 - December 1986
. . . Income2 = Monthly data: October 1986 - March 1987
. . Expenses = Yearly data: 1986 - 1986
. . Interest = PREVIOUS(Cash;0,1) * MONTHLY(YEARLY(Interest Rate))
. . . Interest Rate = 0.08
```

Figure 9.2: Roots view of Mr. Pizza cash flow model

Accumulation Functions

■ Although PREVIOUS and NEXT are sufficient to build practically any time-series calculation, some operations occur so often that Javelin has functions to do them. In particular, summing up the value of a series over time happens so frequently that there are several special functions to do that.

The CUM function (*cumulative*) keeps a running sum of the values of the input series, so that the result at the first period is the same as the input at the first period, the result at the second period is the sum of the first and second periods of the input, the result at the third period is the sum of the first three periods of the input, and so forth. The result of CUM has the same period and time span as its input. The following two formulas compute the same result:

```
Cash = CUM(Cash Flow)
Cash = Previous(Cash;0) + Cash Flow
```

	Cash	Cash Flow	Total Income	Expenses	Interest
Jan 1986	$0.00	$0.00	$10,000.00	$10,000.00	$0.00
Feb 1986	$1,000.00	$1,000.00	$11,000.00	$10,000.00	$0.00
Mar 1986	$3,006.67	$2,006.67	$12,000.00	$10,000.00	$6.67
Apr 1986	$6,026.71	$3,020.04	$13,000.00	$10,000.00	$20.04
May 1986	$10,066.89	$4,040.18	$14,000.00	$10,000.00	$40.18
Jun 1986	$15,134.00	$5,067.11	$15,000.00	$10,000.00	$67.11
Jul 1986	$21,234.90	$6,100.89	$16,000.00	$10,000.00	$100.89
Aug 1986	$28,376.46	$7,141.57	$17,000.00	$10,000.00	$141.57
Sep 1986	$36,565.64	$8,189.18	$18,000.00	$10,000.00	$189.18
Oct 1986	$46,309.41	$9,743.77	$19,500.00	$10,000.00	$243.77
Nov 1986	$57,218.14	$10,908.73	$20,600.00	$10,000.00	$308.73
Dec 1986	$69,299.59	$12,081.45	$21,700.00	$10,000.00	$381.45
Jan 1987					$462.00

Table 9.4: Worksheet with results of cash flow model

Note that the second version says that Cash at a given date is the previous month's Cash added to the current month's Cash Flow, but if it can't compute that (in the first period, because there is no previous month's Cash), it adds the current Cash Flow to 0. The first version is easier to understand and also happens to be faster to compute.

To modify the model to use it, type

Cash = CUM(Cash Flow)

Recalculate the model; the figures for Cash Flow don't change. Here are the functions you can use to sum the data for different periods:

- CUM(expression): Computes the cumulative value of a time series.
- YTD(expression): Computes the year-to-date sum of a time series.
- QTD(expression): Computes the quarter-to-date sum of a time series.
- MTD(expression): Computes the month-to-date sum of a time series.

YTD, QTD, and MTD compute a running sum as CUM does, except that each starts the sum over at the beginning of each year, quarter, or month, respectively. The result of any of these has the same period and time span as its input. When converting the results of these functions to a larger period, you almost always want to set the variable's conversion method to Last Period. Table 9.5 gives a simple example of the results of these functions accumulating the value of Ones, a daily variable that equals 1 each day.

These functions operate on a time series and return a time series. If you want Javelin to look across a time series and give you a single piece of information about that time series (like the sum, maximum, minimum, or average of the whole series), use the following functions (called *T-functions*):

- TSUM(expression): Calculates the sum of an entire time series.
- TAVG(expression): Calculates the average of an entire time series.
- TMAX(expression): Calculates the maximum of an entire time series.
- TMIN(expression): Calculates the minimum of an entire time series.

Any result of these functions is a constant, even though the input is a time series. Since they operate on entire series, TAVG, TMIN, and TMAX disregard input values of #NA (where you haven't typed in data), #BLANK, or

#RANGE (where a formula's input data don't exist). If all the numbers in the input series are #NA or #RANGE, the result is #NA, since Javelin cannot compute an answer from no data.

For example, suppose you wanted to know the overall expenses to expect during the course of your model. You would use TSUM:

Overall Expenses = TSUM(Total Expenses)

To find your maximum and minimum monthly expenses:

Maximum Monthly Expense = TMAX(Total Expenses)
Minimum Monthly Expense = TMIN(Total Expenses)

	Ones	MTD	QTD	YTD
1 Jan 86	1	1	1	1
2 Jan 86	1	2	2	2
...				
30 Jan 86	1	30	30	30
31 Jan 86	1	31	31	31
1 Feb 86	1	1	32	32
2 Feb 86	1	2	33	33
...				
30 Mar 86	1	30	89	89
31 Mar 86	1	31	90	90
1 Apr 86	1	1	1	91
2 Apr 86	1	2	2	92
...				
30 Dec 86	1	30	91	364
31 Dec 86	1	31	92	365
1 Jan 87	1	1	1	1
2 Jan 87	1	2	2	2

Ones is a daily series of 1's

Table 9.5: YTD, QTD, and MTD functions

And similarly for average monthly expenses:

```
Average Monthly Expense = TAVG(Total Expenses)
```

Let's take another look at how Cash is computed:

```
Cash = CUM(Cash Flow)
Cash Flow = Total Income - MONTHLY(Expenses) + Interest
Interest = PREVIOUS(Cash;0) * MONTHLY(YEARLY(Interest
           Rate))
```

Ignoring Total Income and Expenses for a moment, the interest earned each month is the monthly value of 8-percent annual interest multiplied by the previous month's Cash. Since Cash depends on Interest, and Interest depends on Cash, this produces what appears to be a *circular* model. The reason it isn't circular is that Interest depends on PREVIOUS(Cash), which is why we call this a *spiral* model. It is important to distinguish between a spiral model, which appears to be circularly defined, and a circular model, which really is. The distinguishing feature is the way Javelin deals with time.

A circular model is one in which a value depends indirectly on itself, making it impossible to compute an answer directly. In this case, Cash depends on Interest, and Interest depends on Cash. You have created a spiral model in which the calculations are not circular, because Interest really depends on the previous period's Cash, not the current period's Cash. If you had left PREVIOUS out of the interest calculation, the model would indeed be circular, and many of the results would have come up as #CIRC. Javelin can handle circular models as discussed in Chapter 11, "Special Values and Diagnosing Problems." However, they are harder to deal with than are spiral models, and you should try to build spiral models instead whenever possible.

The @ Operator and the VALUE Function

■ One very useful Javelin construct is the @ ("at" sign) operator, which allows you to give a series a value at a particular date. It is useful for "priming" a series with a particular value. For example, say that you started your

pizza business with an initial investment of $3000 in December 1985. You could modify your Cash Flow formula to be

Cash Flow = Total Income - Expenses + Interest; 3000 @ Dec 1985

By itself the @ operator is not very useful, but in combination with the semicolon and functions such as PREVIOUS and GROW it is extremely handy. For example, you can use the @ and semicolon operators to simulate an input variable, such as

Three Points = 100 @ Jan 87; 200 @ Mar 87; 300 @ May 87

Three Points is a series defined at three points. If you want to use one value of a series, no matter what the current date is, you need the VALUE function:

- VALUE(expression, date): Takes a value from a time series and returns it as a constant.

This function is useful mostly to normalize a time series. For example, in

Normalized Price = Price / VALUE(Price, Jan 1984)

Normalized Price is the price relative to the price in January 1984.

One respect in which your model is not very realistic is the estimate of Income and Expenses. You just typed them in as time-series data, growing at arbitrary rates (and easy to enter). Let's be more realistic and assume some base values and growth rates rather than a particular series of data. Javelin's extrapolation and interpolation functions help you do this.

Extrapolation and Interpolation Functions

■ These two functions invent data based on input data. Used judiciously, they can be useful to fill gaps in your data. Used unwisely, they can create large amounts of completely invalid data.

■ *Extending a Time Series Using Extrapolation*

The GROW function is used for extrapolation:

- GROW(expression, rate): Extrapolates from a time series; all existing data in the input time series become the corresponding values of the result. Past the end of the input, it extrapolates from the last period of the input, increasing (or decreasing) the result by the growth rate for each period.

For example, if you assume that Income will be $10,000 in January 1986 and that it will grow at 50 percent per year, you could replace the Income input data with a formula. Type

```
Income = GROW(10000 @ Jan 1986, MONTHLY(YEARLY(Income
         Growth)))
Income Growth = 0.50
```

Don't forget to recalculate the model. In this example, your input time series is defined in only one period, January 1986. The characters @ Jan 1986 identify the series as monthly, since this is a monthly date; as with bank interest, you convert the growth rate to monthly so that you grow at 50 percent a year and not per month. Although the input can be any time series, it is most often of the form *value @ date* as in the example. The result of GROW has the same period as the input expression, but the time span runs from the beginning of the input expression through the end of the default time span.

A note on interest rates: MONTHLY(YEARLY(Income Growth)) will divide Income Growth by 12 to get a monthly growth rate. This is not strictly correct, since it does not take into account the effects of compounding. The correct formula for converting an annual 50-percent rate to a monthly rate is

```
(1 + 0.50) ^ (1 / 12) - 1
```

You can use it in your models if you want, although the simpler and less correct method of dividing by 12 is widely used. The simpler method overstates the growth rate, producing in this case an annual rate of about 63 percent rather than 50 percent. Note that January 1987's Income is 16,320.94 rather than the 15,000 you would expect from 10,000 growing by 50 percent.

■ *Filling Gaps in a Time Series Using Interpolation*

If you have some but not all data points in a series, you can use INTERP to interpolate reasonable values for the missing data:

- INTERP(expression): Replaces any values of #RANGE, #BLANK, or #NA in the result by straight-line interpolation from the nearest numeric values.

For example, if your expenses in January 1986 are $11,000 and are $14,000 in January 1987, you can replace Expenses data with the following formula, by typing

Expenses = INTERP(11000 @ Jan 1986; 14000 @ Jan 1987)

The argument to interpolate can be any time-series expression. That is, you could have a monthly input variable (say Expense Data) with values for January 1986 and January 1987, and write the above formula as

Expenses = INTERP(Expense Data)

If you have only two or three data points, the most common way to use INTERP is to put the data right into the formula. The result of INTERP has the same period and time span as its input. INTERP is very slow if it has to interpolate between input points that are many periods apart. Table 9.6 gives examples of GROW and INTERP.

Creating a More Complex Cash Flow Model

■ Let's expand the Mr. Pizza model to look at a considerably more complex situation. Imagine you are planning to open a chain of pizzerias. Your plans are quite aggressive—you expect to open 32 new stores during the year, with monthly numbers as in Table 9.7.

Starting up a store requires spending $25,000 two months before the store opens and $50,000 one month before it opens. Once a store is open it takes in $3000 per day and costs $2500 per day to operate.

You have a line of credit with the bank, which will lend you money from month to month at an interest rate of 10 percent, and you are required to keep a cash balance of at least $1000. They still pay 8 percent on cash balances. In each period, if you have insufficient cash on hand, you borrow enough to attain the necessary minimum balance, and if you have extra cash, you use it to pay back outstanding loans. Since you can't always predict finances to the penny, allow a 10-percent "fudge" factor in taking and repaying loans. That is, if you borrow to make the minimum balance, you borrow 10 percent more than you expect to need. And if you have extra cash to use to repay the loans, pay back 10 percent less than you expect to have.

Formulas for the full model are shown in Figure 9.3. Rather than try to digest it in one lump, we will go through each section separately. This will also let us introduce the rest of Javelin's time-handling functions. Though this model also runs from January to December 1987, the model's default date

	GROW		INTERP
Formula	CPI = GROW(100 @ 1967, .06)		Overhead = INTERP(100 @ Jan 1987; 900 @ Jun 1987)
Explanation	Grow at 6% a year starting in 1967 at 100.		Overhead is 500 in January and 900 in June. Interpolate the months in between.
Values	1967	100.00	Jan 87 500
	1968	106.00	Feb 87 580
	1969	112.36	Mar 87 660
	1970	119.10	Apr 87 740
	.	.	May 87 820
	.	.	Jun 87 900
	.	.	
	1986	302.56	

Table 9.6: GROW and INTERP functions

range is October 1986 to December 1987, since the expenditures for the restaurants that open in January start the previous November, and some functions, using PREVIOUS, refer back to October.

The first few lines resemble those in the previous model. Cash is still the running sum of Cash Flow but now with Loans Taken added and Loans Repaid subtracted. Also, ;0 is at the end of the formula, which means that in periods where the CUM expression has no value, Javelin assumes that it is zero. In this case, it makes the cash in October 1986 zero, which you need to be able to compute the interest for November.

Let us consider the operating cash flow and the loan cash flow separately. In Figure 9.4 you see the roots of Cash Flow. Use the CUM function again to compute the number of stores, by adding up the number opened each month.

Functions That Count Days in Larger Periods

■ There is one new function, MDAYS; use it to compute Income. Income is $3000 per day, meaning that you must compute monthly sales as a function of the number of days in a month. From previous experience converting yearly interest rates to monthly ones, you could say

Income = MONTHLY(DAILY(3000) * Stores)

New Stores = < Monthly data: Nov 1986 – Feb 1988 >

Nov 1986 0	May 1987 4	Nov 1987 0
Dec 1986 0	Jun 1987 4	Dec 1987 0
Jan 1987 3	Jul 1987 4	Jan 1988 0
Feb 1987 3	Aug 1987 4	Feb 1988 0
Mar 1987 3	Sep 1987 4	Mar 1988 .....
Apr 1987 3	Oct 1987 0	Apr 1988 .....

Table 9.7: Number of new stores opened

This would work, but MDAYS gives Javelin a much faster way of doing it. MDAYS returns the number of days in each month, so that you can compute the monthly income based on revenues of $3000 a day per store.

Javelin 1.1 has no functions that perform date arithmetic per se, but you can do many things you'd want to do with date arithmetic with combinations of these functions. Javelin PLUS has date variables and arithmetic, discussed in Chapter 20, "Strings and Dates." Here are the functions that count days:

- YDAYS( ): Returns a yearly time series containing the number of days in each year.
- QDAYS( ): Returns a quarterly time series containing the number of days in each quarter.
- MDAYS( ): Returns a monthly time series containing the number of days in each month.

YDAYS( ) is equivalent to YEARLY(DAILY(1)), but it is considerably faster and more concise. QDAYS( ) and MDAYS( ) are equivalent to QUARTERLY

```
Cash = CUM(Cash Flow + Loans Taken - Loans Repaid);0
Cash Flow = Income - Total Expenditures + Deposit Interest -
     Loan Interest
Daily Operating Expense = 2500
Deposit Interest = PREVIOUS(Cash) * Deposit Interest Rate
Deposit Interest Rate = 0.07 / 12
Fudge = 0.1
Income = 3000 * MDAYS() * Stores
Loan Balance = ROUND(CUM(Loans Taken - Loans Repaid),2)
Loan Interest = PREVIOUS(Loan Balance) * Loan Interest Rate
Loan Interest Rate = 0.1 / 12
Loans Repaid = MIN(IF(PREVIOUS(Cash) + Cash Flow > Minimum Balance,
     (PREVIOUS(Cash) + Cash Flow) * (1-Fudge),0),PREVIOUS(Loan
     Balance));0
Loans Taken = IF(PREVIOUS(Cash) + Cash Flow < Minimum Balance,(
     Minimum Balance - PREVIOUS(Cash) - Cash Flow) * (1 + Fudge),0)
     ;0
Max Loan Balance = TMAX(Loan Balance)
Minimum Balance = 1000
New Stores = < Monthly data: November 1986 - February 1988 >
Operating Expenditures = Stores * MDAYS() * Daily Operating
     Expense
Startup Expenditures = 50000 * NEXT(New Stores,1) + 25000 * NEXT(New
     Stores,2)
Stores = CUM(New Stores)
Total Expenditures = Operating Expenditures + Startup
     Expenditures
```

Figure 9.3: More complex cash flow model

(DAILY(1)) and MONTHLY(DAILY(1)). These functions are useful mostly for dealing with data that arrive in a period other than the most natural one. None of these functions takes any arguments (nothing goes inside the parentheses) and each returns values for the model's default time span, converted into the appropriate time period.

■ *Continuing the Enhanced Mr. Pizza Model*

Operating Expenditures is computed the same way as Income, based on spending $2500 a day per store. Startup Expenditures uses the NEXT function to look ahead at how many stores will be opened one and two months hence. The second argument to NEXT is the number of periods ahead to look, so the formula says that you are spending $50,000 for each store that will open one month hence and $25,000 for each store that will open two months hence. Notice that in New Stores (in Table 9.4) there are values back to Nov 1986 and ahead to Feb 1988. The NEXT functions in Startup Expenditures look up to two months ahead, so that you need New Stores values until Feb 1988 to enable the NEXT function to compute a value for Dec 1987.

Table 9.8 shows some monthly cash flow values. Notice in the last line that the Combine method for Cash is set to Average, since it represents a stock rather than a flow.

Now turn your attention to the loans, as shown in Figure 9.5. Loan Balance is the running sum of Loans Taken less Loans Repaid.

```
Cash Flow = Income - Total Expenditures + Deposit Interest -
    Loan Interest
. Income = 3000 * MDAYS() * Stores
. . Stores = CUM(New Stores)
. . . New Stores = < Monthly data: Nov 1986 - Feb 1988 >
. Total Expenditures = Operating Expenditures + Startup
    Expenditures
. . Operating Expenditures = Stores * MDAYS() * Daily Operating
    Expense
. . . Stores = CUM(New Stores)
. . . Daily Operating Expense = 2500
. . Startup Expenditures = 50000 * NEXT(New Stores,1) + 25000 * NEXT(
    New Stores,2)
. Deposit Interest = PREVIOUS(Cash) * Deposit Interest Rate
. . Cash = CUM(Cash Flow + Loans Taken - Loans Repaid);0
. Loan Interest = PREVIOUS(Loan Balance) * Loan Interest Rate
. . Loan Balance = ROUND(CUM(Loans Taken - Loans Repaid),2)
. . Loan Interest Rate = 0.1 / 12
```

Figure 9.4: Roots view of Cash Flow

Loans Taken is defined by a formula that is long but straightforward. It asks whether your current cash flow will drop below the minimum balance. If so, you need to borrow the difference, adjusted by the fudge factor; if not you needn't borrow anything. A familiar ;0 at the end provides zeros in the first period so spiral formulas that depend on Loan Balance can get started. Loans Repaid performs a similar calculation in reverse. If you go above the minimum balance, use the extra to repay loans, adjusted again by the fudge factor. A MIN function keeps you from repaying more than the outstanding loan balance.

	Cash	Cash Flow	Income	Total Expenditures
Oct 86	–	–	–	
Nov 86	8600.00	(75,000.00)	–	75,000.00
Dec 86	22,804.65	(225,646.50)	–	225,000.00
1986	10,468.22	(300,646.50)	–	300,000.00
Jan 87	16,925.77	(181,062.40)	279,000.00	457,500.00
Feb 87	13,913.08	(145,056.56)	504,000.00	645,000.00
Mar 87	11,284.48	(115,757.83)	837,000.00	947,500.00
Apr 87	12,593.14	(126,215.91)	1,080,000.00	1,200,000.00
May 87	5767.78	(59,270.98)	1,488,000.00	1,540,000.00
Jun 87	1298.01	(7747.84)	1,800,000.00	1,800,000.00
Jul 87	6549.68	64,198.77	2,232,000.00	2,160,000.00
Aug 87	23,327.03	226,720.63	2,604,000.00	2,370,000.00
Sep 87	49,789.51	474,568.03	2,880,000.00	2,400,000.00
Oct 87	324,186.03	494,456.60	2,976,000.00	2,480,000.00
Nov 87	806,077.12	481,891.09	2,880,000.00	2,400,000.00
Dec 87	1,306,779.23	500,702.12	2,976,000.00	2,480,000.00
1987	214,874.24	1,607,425.74	22,536,000.00	20,880,000.00

Table 9.8: Worksheet of cash flow values

Table 9.9 shows the loan numbers. You take out loans in the early months. As stores open and revenues increase, you pay them back until in October 1986 you have repaid them fully. The yearly 1986 numbers show the total amount borrowed and repaid during the year and the interest paid on the loans and collected on the cash balances. Since Loan Balance is a stock, you've set its combine method to Last Period, so you see that the loan balance at the end of the year is zero. At the bottom of the worksheet is an exception cell showing Max Loan Balance, which uses the TMAX function to compute the largest outstanding loan and hence the size of the credit line needed.

This is the basic outline used in all cash flow models. In a real model there are many more categories, such as taxes and overhead, and the calculations of income and expenses usually go into more detail, but the structure of the model remains the same.

■ *Adding a Few More Changes*

You could model the income from the stores more accurately. Rather than assuming an income of $3000 every day, for Fridays and Saturdays set the income at $3500 and for Sundays only $2000.

The changes to the model are shown in Table 9.10. You have a new variable called Daily Sales, which shows the day-by-day sales pattern, as described above. Rather than approximating 3000 * MDAYS() for the monthly sales per store, use MONTHLY(Daily Sales), which adds up the daily numbers for each month (using the Sum conversion) to get the monthly sales per store.

```
Loan Balance = ROUND(CUM(Loans Taken - Loans Repaid),2)
. Loans Taken = IF(PREVIOUS(Cash) + Cash Flow < Minimum Balance,
     (Minimum Balance - PREVIOUS(Cash) - Cash Flow) * (1+Fudge),0);0
. . Cash = CUM(Cash Flow + Loans Taken - Loans Repaid);0
. . Cash Flow = Income - Total Expenditures + Deposit Interest -
       Loan Interest
. Loans Repaid = MIN(IF(PREVIOUS(Cash) + Cash Flow > Minimum
     Balance, (PREVIOUS(Cash) + Cash Flow) * (1 - Fudge),0),PREVIOUS
     (Loan Balance));0
. . Minimum Balance = 1000
. . Fudge = 0.1
```

Figure 9.5: Roots view of outstanding Loan Balance

Let's look at the definition of Daily Sales in more detail (see Figure 9.6). It consists of three parts connected by semicolons. The first is Daily Sales Pattern, an input variable that contains the daily numbers for the first seven days of January 1986, shown in the first data column of the worksheet in Table 9.10. The next is a PREVIOUS function, which says that the value, other than where it is defined by Daily Sales Pattern, is the same as the value seven days earlier. This makes the values repeat each week. The third is the familiar ;0, which makes the values before the beginning of 1987, when you have no sales anyway, zero.

	LOANS			INTEREST	
Date	**Taken**	**Repaid**	**Loan Balance**	**Loan**	**Deposit**
Oct 86	–	–	–		
Nov 86	83,600.00	–	83,600.00	–	–
Dec 86	239,851.15	–	323,451.15	696.67	50.17
1986	323,451.15	–	323,451.15	696.67	50.17
Jan 87	175,183.52	–	498,634.67	2695.43	133.03
Feb 87	142,043.86	–	640,678.53	4155.29	98.73
Mar 87	113,129.23	–	753,807.76	5338.99	81.16
Apr 87	127,524.57	–	881,332.33	6281.73	65.83
May 87	52,445.62	–	933,777.95	7344.44	73.46
Jun 87	3278.06	–	937,056.01	7781.48	33.65
Jul 87	–	58,947.10	878,108.91	7808.80	7.57
Aug 87	–	209,943.28	668,165.63	7317.57	38.21
Sep 87	–	448,105.55	220,060.08	5568.05	136.07
Oct 87	–	220,060.08	–	1833.83	290.44
Nov 87	–	–	–	–	1891.09
Dec 87	–	–	–	–	4702.12
1987	613,604.86	937,056.01	–	56,125.61	7551.34
			Maximum Loan Balance =	937,056.01	

Table 9.9: Monthly loan numbers

Time-Span Limiting Functions

■ In Figure 9.7 is a slightly different way to get the same effect. In this case Daily Sales is defined by a four-part semicolon set:

```
1) RANGE(3 Jan 1986, 4 Jan 1986, 3500);
2) 2000 @ 5 Jan 1986;
3) PREVIOUS(Daily Sales, 7);
4) 3000
```

The RANGE function is new, and one of a number of functions Javelin has for limiting a time series. Let's look at them before picking apart the Daily

Day	Pattern	Daily Sales	Month	Stores	Income
1 Jan 87	3000	3000	Jan 87	3	280,500
2 Jan 87	3000	3000	Feb 87	6	504,000
3 Jan 87	3500	3500	Mar 87	9	832,500
4 Jan 87	3500	3500	Apr 87	12	1,080,000
5 Jan 87	2000	2000	May 87	16	1,504,000
6 Jan 87	3000	3000	Jun 87	20	1,780,000
7 Jan 87	3000	3000	Jul 87	24	2,232,000
8 Jan 87		3000	Aug 87	28	2,604,000
9 Jan 87		3000	Sep 87	32	2,880,000
10 Jan 87		3500	Oct 87	32	2,992,000
11 Jan 87		3500	Nov 87	32	2,864,000
12 Jan 87		2000	Dec 87	32	2,976,000

Table 9.10: Using a weekly income pattern

Sales formula. Time-span limiting functions limit the time span of expressions. The usual reason to use them is that you have a situation in which an activity starts or stops on a given date.

- RANGE(start date, end date, expression): Makes a time series that starts and ends at the given dates.

The two dates must be of the same period (both days, months, or such), and the end date must be after the start date. If the expression is a constant, the result is a series with the constant's value at each point. If the expression is a series, it is shortened as needed to accommodate the given start and end dates. RANGE will not convert time periods nor invent new data, which means that the dates must have the same period as the expression. If the expression starts after the given start date or ends before the given end date, the result has the start or end date, respectively, of the expression.

This means that the RANGE function here defines a two-day series, starting on 3 January 1986 and ending on 4 January 1986. During each day in this range, the function's value is 3500. Another way to say this would be

3500 @ 3 Jan 86; 3500 @ 4 Jan 86

The BEGIN and END functions work similarly to RANGE, but they define only one end of the series.

- BEGIN(date, expression, optional number of periods): Makes a time series based on the expression starting at the given date.
- END(date, expression, optional number of periods): Makes a time series based on the expression ending at the given date.

The optional number of periods says how long the series should be. If you don't specify the number of periods, BEGIN goes on to the end of the model's default time span and END goes back to the beginning of its time span.

```
Income = MONTHLY(Daily Sales) * Stores
. Daily Sales = Daily Sales Pattern; PREVIOUS(Daily Sales,7); 0
. . Daily Sales Pattern = < Daily data: 1 January 1987 -
      7 January 1987 >
. Stores = CUM(New Stores)
. . New Stores = < Monthly data: November 1986 - February 1988 >
```

Figure 9.6: Weekly cyclic data

If the expression is a constant, the result is a time series with the value of the constant at each point in the series (like RANGE). If the expression is a time series, BEGIN and END "snip off" the series at the date you specify. The resulting time series, for BEGIN, starts at the given date and ends the number of periods later (if given), or otherwise at the end of the default time span. The result series for END ends at the given date and starts the number of periods back (if given), or otherwise from the beginning of the default time span.

BEGIN and END will not convert time periods nor make a time series longer than it already is, so the given date must have the same period as the expression. If the given date is outside the time span of the expression, the result still has the same starting or ending date as the expression. Similarly, if the number of periods would make the series longer than the expression, the result ends or starts at the same date as the expression.

Getting back to the formula for Daily Sales, its second part is an @ operator that sets the value for the first Sunday in 1986, the third part is a PREVIOUS function that makes the values repeat weekly, and the last is 3000, to set all the days' values not otherwise defined at $3000. The PREVIOUS operator must come before the 3000; if they were in the other order all values except for the three defined by RANGE and @ would be 3000, because a semicolon takes the expression to its left in preference to the expression to its right.

There are three more groups of functions you have not used yet, but which are important aspects of dealing with time. We'll finish this chapter by discussing them.

Smoothing Functions

■ Frequently you want to smooth out your data by combining the values from several periods. Javelin provides functions to calculate running sums, averages, maximums, and minimums. There are also more sophisticated functions

```
Income = MONTHLY(Daily Sales) * Stores
. Daily Sales = RANGE(3 Jan 1987, 4 Jan 1987, 3500);
  2000 @@ 5 Jan 1987; PREVIOUS(Daily Sales,7);3000
. Stores = CUM(New Stores)
. . New Stores = < Monthly data: November 1986 - February 1988 >
```

Figure 9.7: Another way to define weekly cyclic data

you can use for smoothing by computing least-squares trend lines, which are discussed in Chapter 15, "Statistics." Here are the smoothing functions:

- RSUM(expression, optional number of periods back): Computes a running sum series.
- RAVG(expression, optional number of periods back): Computes a running average series.
- RMAX(expression, optional number of periods back): Computes a running maximum series.
- RMIN(expression, optional number of periods back): Computes a running minimum series.

If the optional number of periods back is specified, the value at each date is the sum, average, maximum, or minimum of that many periods' data. For example, if the number of periods is 3, the value of the result at October will be computed from the expression for August, September, and October. If there are not enough previous periods' data, the value is computed from the beginning of the expression. In the example above, if data for the expression begins in March, the result for April is computed from March and April.

If the optional number of periods back is omitted, the value at each date is the sum of the expression from that date back to the beginning of the input series. That makes these functions the same as their T-function cousins, TSUM, TAVG, TMAX, and TMIN.

Input values of #NA, #BLANK, or #RANGE are disregarded when computing averages, maxima, or minima. The resulting time series of these functions has the same period and time span as the expression.

For example, to compute running three-month sales averages, type

```
Smoothed Sales = RAVG(Sales, 3)
```

If you had a series of weekly stock prices and wanted to compute the running 52-week highs and lows, you would type

```
52 Week High = RMAX(Stock Prices, 52)
52 Week Low = RMIN(Stock Prices, 52)
```

Rate Functions

■ Another common calculation is the relative or absolute change from one period to another. Javelin has three functions to help with this:

- CHANGE(expression): For each period, computes the change from one period to the next.
- PCT(expression): For each period, computes the relative change from one period to the next.
- GROWRATE(expression): Computes the compound growth rate of a time series.

For each period, the result of CHANGE at a period is the difference between the input value at that period and the input value at the previous period. Obviously, this means that CHANGE(Sales) is the same as Sales – Previous(Sales), but CHANGE is clearer and more efficient.

Similarly, the value of PCT at a period is the relative change, that is, the change divided by the previous value. If the previous value of the input was 100 and the current value is 150, the CHANGE is 50, and the PCT is 50 percent or .5. If you are selling pizzas and want to see the month-to-month change in sales, you would type

```
Pizza Sales Increase = CHANGE(Pizza Sales)
```

To compute the relative change in pizza sales, you would type

```
Pizza Sales Pct Increase = PCT(Pizza Sales)
```

or

```
Pizza Sales Pct Increase = (Pizza Sales - PREVIOUS(Pizza
                           Sales))/PREVIOUS(Pizza Sales)
```

Again, the PCT form is easier to read and faster to calculate.

The name PCT is rather a misnomer, since the result is really a fraction rather than a percentage. You will usually want to set the numeric format of the result of a PCT function to Percent display on the Variable settings sheet, so that the numbers look like percentages.

The resulting time series of CHANGE or PCT has the same period as the input expression, but it starts one period later than the input. See Table 9.11 for an example.

GROWRATE takes the first and last values of the input time series and figures what periodic growth rate would have been needed to grow the first value to the last value over the time span of the series. You would type

Pizza Growth Rate = GROWRATE(Pizza Sales)

If your Pizza Sales were 100 in January and 250 in June, GROWRATE would compute your average monthly growth rate as .2011, or 20.11 percent (see Table 9.11).

If you were running a telephone hotline and tracking the number of calls per year, and in your first year you received 50,000 calls and in your fourth year 80,000 calls, you could compute your growth rate by typing

Call Growth Rate = GROWRATE(Yearly Calls)

The result is .1696, meaning an annual growth rate of 16.96 percent. The result of GROWRATE is a constant, even though its input is a time series.

Date Arithmetic Functions

■ Finally, there is a group of functions that allows you to do some date arithmetic, even in Javelin 1.0 and 1.1 (without date variables):

- SINCE(date): Returns a time series of the number of periods after a given date.
- UNTIL(date): Returns a time series of the number of periods before a given date.
- BETWEEN(date1, date2): Calculates the number of periods between two dates.

- COUNT(time series): Counts the number of periods of which the series has a value. Periods with values of #NA, #BLANK, and #RANGE are not counted.
- LENGTH(time series): Counts the number of periods for which the series is defined, including #NA, #BLANK, and #RANGE values between the first and last data points.

SINCE returns a series that counts up, and UNTIL returns a series that counts down. In both cases, the value is zero at the specified date (or would be if the date is outside the default date range). These are the most useful functions for performing date arithmetic, since they give you a series of numbers that you can manipulate and that correspond directly to dates. For example, assume that you have a series of payments due each year from 1987 to 1995, and you want to open a set of term accounts in 1986 at a bank that will mature yearly. Assuming a fixed annual interest rate, how much will you have to put in each account to cover the various payments? The interest-rate formula is

```
(1 + rate) ^ N
```

where N is the number of periods for which interest is earned. You can use SINCE to generate the right values of N, so that the amounts to deposit are

```
Term Deposits = Payments Due / (1 + Rate) ^ SINCE(1986)
```

These functions take a date as an argument and return values for the model's default time span, converted into the appropriate time period.

	Jan 87	Feb 87	Mar 87	Apr 87	May 87	Jun 87
Pizza Sales	100	150	130	170	200	250
Sales Increase (CHANGE)	50	−20	40	30	50	
Sales Pct Increase (PCT)		50.0	−13.3	30.8	17.6	25.0
			Growth Rate = 20.11%			

Table 9.11: CHANGE, PCT, and GROWRATE functions

BETWEEN(Jan 86, Mar 86) is 2, since there are two months from January to March, and BETWEEN(1950, 2000) is 50, since there are 50 years from 1950 to 2000. BETWEEN is most useful in computing the time periods for BONDPRICE and BONDYIELD and for computing interest for single investments. For example, assume that your father banked $100 for you in May 1954 at 3-percent annual interest, compounded monthly. Disregarding changes in the interest rate, to see how much your account is worth in January 1987, type

```
Bank Account = 100.00 * (1 + .03 / 12) ^ BETWEEN(May 1954, Jan
               1987)
```

Your account has grown to $266.12. The two dates given must be of the same period. The result of BETWEEN is a constant.

Conclusion

■ Javelin's ability to work with time sets it apart from spreadsheet programs. The wide variety of time-related functions is a little mind-boggling, but you will find yourself using a surprisingly large number as your models grow more complex.

Next, we will look at custom time periods, and how to define and use them.

CHAPTER TEN

Custom Time Periods

In Chapter 9, we described how Javelin handles time, especially in converting between time periods. But if the time periods Javelin provides won't work for your model, Javelin lets you define *custom periods* of your own.

There are two different cases in which custom periods will help you: when you need a nonstandard time period and when you have data not at all related to time. By using the command / **D**efine **T**ime period and the Custom Period settings sheet, you can specify the name of the period as well as how it relates to normal time, if at all. For example, if your company has a factory that works a three-shift schedule, you can define a custom period called Shift that relates to days, with three shifts per day. You can then use Shift as a period when entering dates, for example, 1 Shift 26 Jan 88, or 3 Shift 12 October 89. Javelin can combine data entered by shift into daily, weekly, monthly, quarterly, or yearly data using the same combining methods you saw in Chapter 9.

Time-Related Custom Periods

■ When you use nonstandard time periods, the data still relate to the calendar, but you organize time differently. Time-related custom periods can be used for

- 5-4-4 production calendars, where the assignment of weeks to months is made in a special way;
- Airline planning calendars, because airlines make their plans in two-month intervals;
- Shifts, where you want days subdivided into shifts;
- Checks, where you want checks grouped by month, but the numbers of checks differ from month to month.

In the first three instances, standard time periods are divided up in regular but nonstandard ways. In the last, time periods are used to track something that isn't exactly chronological. Checks are partially chronological in that each has a date, but several checks can be written on one date in any order (it doesn't matter which you process first, although for convenience you probably sort them by check number).

Nontime Custom Periods

■ In other models, data may not be chronological at all. In this case you need to define a *nontime* custom period, using some other dimension, such as

- Employee number
- PC number
- Invoice number
- Department number
- Chapter number

or any other data to which you can assign a number.

Imagine creating a compensation model of your 312 employees. You might call your custom period Emp and have it run from 1 Emp to 312 Emp. If you define a custom period with no relation to time, the periods start at 1 and end wherever you want, with an upper limit of about 65,000.

You can even use nontime custom periods for time that isn't connected to the calendar. For instance, say you are managing a project but don't know yet when it will start. You might decide that the project will run for 40 weeks, whenever it starts. In that case, you could define a period called ProjWeek, and the periods in your model would run from 1 ProjWeek to 40 ProjWeek.

Defining a Custom Period

■ Throughout Javelin, you enter dates—starting dates, ending dates, and others. Whenever you are prompted for a date, you can use a normal date (e.g., 1 Feb 87 or Nov 89) or a date with a custom period (e.g., 3 Shift 24 March 88 or 16 Emp). To Javelin, these are all dates, since they are expressed using a time period, whether normal or custom.

■ A, B, and Record Periods

All Javelin models have three custom periods, which are initially called *A*, *B*, and *Record* (in Javelin 1.1, models have only the A and B custom periods).

In most models, these time periods remain unused and therefore undefined. One way to use a custom period in a model is to start using A, B, or Record in your dates. The easiest way to do this is in the Worksheet view. Start with a blank model and view a worksheet in the bottom window with the command / **V**iew **W**orksheet. Call the worksheet Usertime.

If you use the A or B time period without defining it, Javelin considers it to be a nontime custom period. When you define a custom period, you tell Javelin if the period relates to a normal time period Javelin knows. Record can be used only as a nontime period—Javelin won't let you relate it to another period.

■ *First Example: 5-4-4 Production Calendar*

Let's use the A period as a nontime period and define it to be a 5-4-4 production week. The 5-4-4 production calendar is a response to the fact that, from one year to another, months contain both different numbers of work days and different numbers of weeks. This makes direct year-to-year comparisons difficult. As a result, many companies have adopted a calendar in which January contains five work weeks, February and March four work weeks, April five, May and June four, and so on to add up to 52 weeks in a year.

Move to the left border on the first row (not the corner), widen the left border column to 30, and type a date:

1A

Move down one row and press ⟵ twice. Javelin fills in entries for 2A and 3A, just as it would for any other dates (see Chapter 3). The same thing works for B period data, but it does not work for just any letter. In the left border, type

1C

Move down and press ⟵: Nothing happens. A and B are special; they are not just letters, they are periods, like months and years. Since C doesn't mean anything to Javelin, it is assumed to be a variable name. Get rid of the 1C label using the Del key.

To define a variable with period A, move to the top border of the first column, and enter the variable name:

Production

Now move down to the first row in the same column and type in some data:

Period	You type
1A	110
2A	150
3A	70

Production is now defined as an input variable with period A. The A period is not yet related to normal time. Try asking Javelin to convert these data to another time period. Move into the left border below the A dates and type a monthly date:

Jan88

■ #NODATE: When Javelin Can't Convert Between Periods

Javelin can't give you an answer for Production for January 1988, because it doesn't know how to convert data from A periods to months. Instead, it displays the special value #NODATE (see Figure 10.1). To fix a #NODATE, define the custom period in terms of a normal time period or don't try to do the conversion.

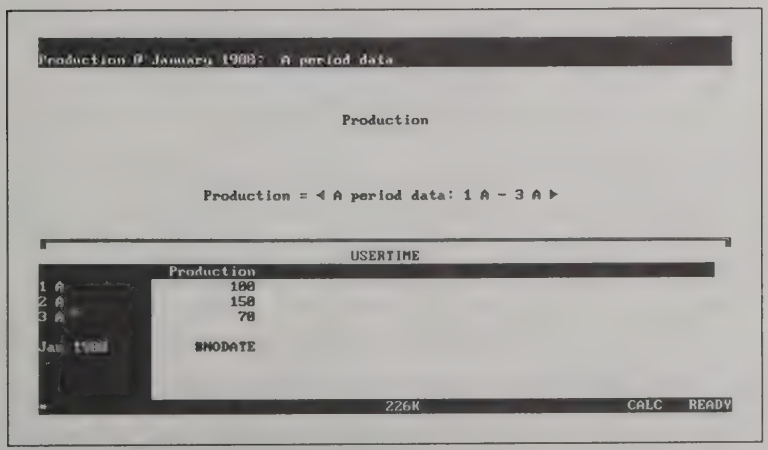


Figure 10.1: Worksheet view with *A* period data

■ Defining a Nontime Custom Period

Working with custom periods called A or B is confusing; it is hard to remember what they stand for. Record is usually used when creating a database (see Chapter 21) and doesn't need to be renamed as often. Instead, you should change their names to something meaningful. To do this, use the Custom Period settings sheet, which you display with the command / **D**efine **T**ime period. Javelin prompts you to type in the existing name of the custom period that you want to define, not the new name you want. Use the Spell key for a list of existing names, as in Figure 10.2. When Javelin starts, the names are always A, B, and Record. Choose A.

Then you see the Custom Period settings sheet, as in Figure 10.3. It contains the two items needed to define a period: the name of the custom period and what other period it relates to, if any. You can rename a custom period at any time. You are using A as a 5-4-4 production week, so change the name to ProdWeek. Custom time period names can be up to 15 characters long.

Custom periods start out not being related to time. The default setting for *Defined in terms of* is *No base period.* Leave it that way for now. Press Esc. As soon as you change the name of the period, the names change on the worksheet too. In fact, they change all over your model.

The default time span for variables using a custom time period is 100 periods, e.g., 1A through 100A.

■ Defining a Time-Related Custom Period

Custom periods allow you to apply Javelin's time-handling features to periods of any length, up to one year. Javelin does this by allowing you to subdivide existing time periods into your own periods. When defining your own period, think about the next bigger time period and how it relates to yours: such as how many shifts in a production week, how many production weeks in a month, how many checks in a month, or how many hours in a business day. Since the largest period Javelin knows about is the year and custom periods must subdivide a normal period, a custom period cannot be larger than a year.

To relate production weeks to real time, you must tell Javelin that ProdWeek is defined in terms of months and specify how many production

weeks are in each month of the time span you plan to use. Give the command / **D**efine **T**ime period ProdWeek and select Month. A full-screen table appears in which you relate ProdWeeks to months by telling Javelin how many ProdWeeks are in each month. Where the 5-4-4 nature of production weeks is reflected, fill out the table as in Figure 10.4.

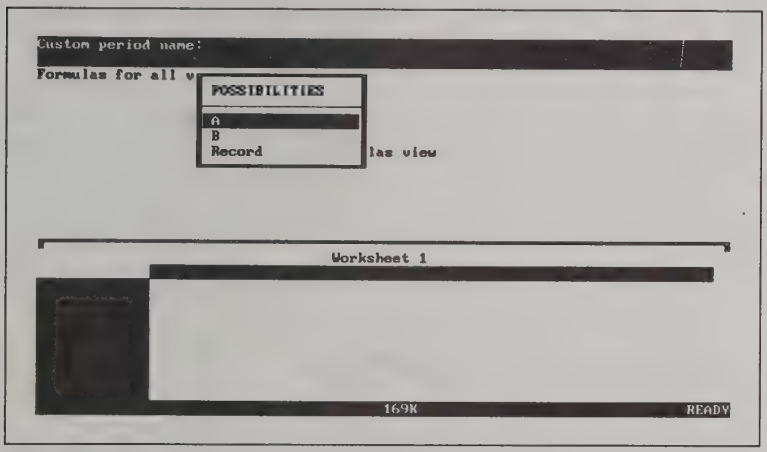


Figure 10.2: Preparing to define a custom period

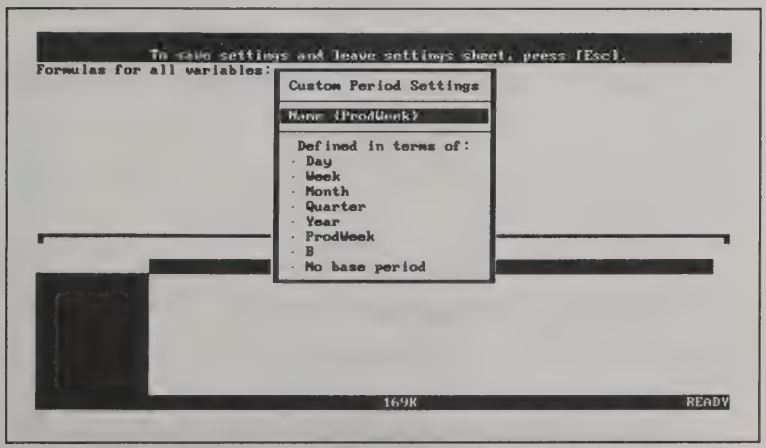


Figure 10.3: The Custom Period settings sheet

To exit from this table, press the Esc key, since Javelin considers this to be part of the Custom Period settings sheet.

Converting To and From Custom Periods

■ Once you have defined your custom period in terms of a time period Javelin knows, data can be converted between the custom period and other time periods. To convert *from* a custom period, Javelin does the following:

- Converts from the custom period to the time period in terms of which it is defined;
- Converts from that regular time period to the period you want.

For example, to convert ProdWeek data to quarterly data, Javelin internally combines four or five ProdWeeks values to get monthly values and then converts the monthly values to quarterly. Converting the other way works the same but in reverse: Javelin first converts quarterly values to monthly values, then divides the monthly values by four or five to get production week values. All time-related settings on the Model and Variable

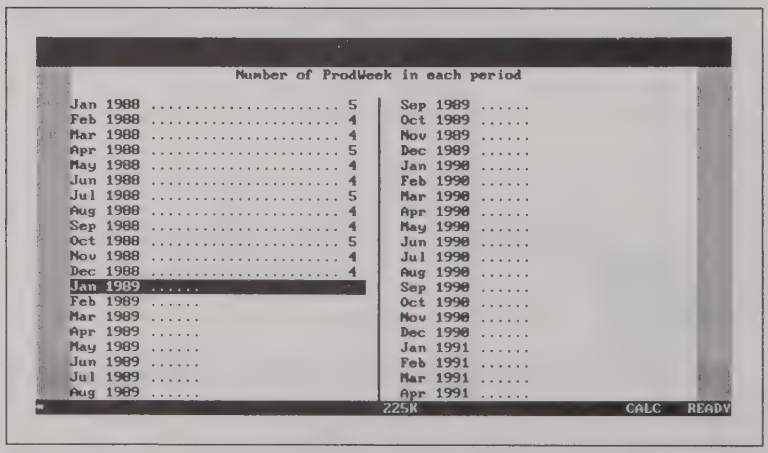


Figure 10.4: Table relating production weeks to months

settings sheets apply to custom periods, to control how Javelin combines smaller periods into larger ones (by summing, averaging, or taking the last period) and splits larger periods into smaller ones (by dividing, interpolating, or repeating). Splitting evenly or by days also applies.

■ Custom Period Conversion Functions

In addition, in formulas you use functions with the names of the time periods to convert data from one time period to another: DAILY converts a variable from whatever its time period is to a daily series. Similarly, when you define a time-related custom period, Javelin defines a new function with the name of the period. When you defined ProdWeek, Javelin also defined a function called PRODWEEK for converting variables into the ProdWeek period. To convert monthly overhead data to ProdWeek, you could use the PRODWEEK function:

```
Overhead Applied to Production = ProdWeek(Overhead) *
                                 Overhead Production Factor
```

For this reason, you should be careful in naming your time-related custom periods. If you name one something like SUM (or any other name that Javelin already uses for a function) *you will not be able to convert to that period in a formula.* That is because Javelin already has a SUM function, which it uses in preference to the custom time period function of the same name. This is an unlikely situation, but it would be confusing.

■ Other Time Functions Used with Custom Periods

The T-functions (TSUM, TAVG, TMAX, and TMIN) also work for variables that are defined using a custom period. For normal time-series variables, T-functions calculate the sum, average, maximum, or minimum value of a variable across time, that is, for all periods for which the variable is defined. With custom periods, they do the same. TSUM of a variable using a custom period is the sum of all the values of the variable. For example, if you have defined a variable Salary that has a nontime custom period Emp (for employee),

```
TMAX(Salary)
```

tells you the highest salary, the largest value across all employees.

Now you have an idea how custom periods work. Let's leave the 5-4-4 production week example (save it for later as 544.MDL) and see how custom periods are actually used in two fairly realistic models. The first one uses a nontime custom period; in it you determine if an employee benefit plan matches a set of regulations. The second one uses a time-related custom period to group checks and deposits in a checkbook.

Employee Model for 401K Plans

■ In a 401K retirement plan, the employees are allowed to contribute some portion of their pay toward a tax-sheltered retirement plan. The government has made many rules about such plans to ensure that they benefit all employees equitably, rather than being skewed mainly to help highly paid employees. The employer can set a maximum percentage that each employee can contribute. Employees are not required to contribute, but the employer can contribute a fixed percentage on each employee's behalf.

Let's set up a model that analyzes a proposed 401K plan to make sure that it is legal. You will test two government rules. The first is that the average percentage of pay contributed by the highest-paid third of the employees cannot be more than 3 percent greater than the average percentage contributed by the rest of the employees. For example, if the lowest paid two-thirds contribute an average of 6 percent of their salaries, the highest paid third cannot contribute more than 9 percent. (These rules change every year, so this one may no longer apply.) The second rule is that the percentage contributed by the highest-paid third cannot be more than 2.5 times the average percentage contributed by the rest.

■ The Custom Period: Emp

Where do custom time periods enter such a model? To find out if a 401K plan conforms to the regulations, you must keep track of several pieces of information about each employee:

- Salary: annual salary;
- Rank: rank by salary, with the highest-paid employee ranked as 1;
- Desired Contribution: amount contributed annually to the plan.

These pieces of information will be your variables (for those familiar with databases, they correspond to fields in a database). An employee number associates each salary with a particular rank and a particular contribution: for example, Employee 1 earns $59,000, ranks ninth, and contributes $0 to the 401K plan; Employee 2 earns $155,000, ranks second, and contributes $7000. Employee Number is the custom period; it associates the instances of variables (just as a record in a database associates the database's fields). Since employee numbers have no relation to time periods, this is a nontime custom period.

■ *Input Variables for 401K Model*

This company has 15 employees. Table 10.1 contains the raw employee data. For each employee there is Salary, Desired Contribution, and Rank, which ranks them in salary order from 1 for the highest paid to 15 for the lowest. In Chapter 13, "Advanced Worksheets and Printing," you'll see how to compute the rank variable automatically, but for the moment assume that you typed it in. The Employee Names are also just label cells in the worksheet. (In Chapter 20, "Strings and Dates," you'll see how to use Javelin PLUS to store the names as text values of a variable.)

To set up this model, define time period A to be called Emp, with no base period. Using / **D**efine **V**ariables, define the input variables Salary, Rank, and Desired Contribution as having period Emp. Then enter the data, in either Worksheet or Table view.

Figure 10.5 shows the five input variables. Along with the three series variables are two constants, the Maximum Contribution that each employee can make, in this case 5 percent, and the Employer Paid Contribution, in this case 6 percent. Enter the formulas as shown for the two constants. Now you are ready to test the plan.

■ *Calculated Variables for 401K Model*

First you must test that the top third contributes no more than 3 percent more than the bottom two-thirds. Figure 10.6 shows the formulas needed to do this. You must know the total number of employees; Total Employees counts the number of salaries you've entered. Enter the formula

```
Total Employees = COUNT(Salary)
```

You must divide the employees into two groups, the Highest Paid third and the Lowest Paid two-thirds. To select the highest-paid employees, define a variable Highest Paid Flag that is 1 for the high third and 0 for the rest (as shown in Table 10.2). This is a *flag* because it "flags" those employees who match.

This would be the most straightforward formula:

Highest Paid Flag = IF(Rank < = (Total Employees / 3), 1 , 0)

	Name	Salary	Rank	Desired Contribution
1 Emp	Bernoff	59,000	9	0
2 Emp	Dewar	155,000	2	7000
3 Emp	Gallant	16,000	15	0
4 Emp	Hall	36,000	11	1000
5 Emp	Homer	68,000	7	6600
6 Emp	Ilkins	60,000	8	6000
7 Emp	Jones	90,000	5	7000
8 Emp	Levine	170,000	1	8000
9 Emp	Matthews	28,000	13	0
10 Emp	O'Rourke	145,000	3	12,000
11 Emp	Preston	24,000	14	650
12 Emp	Rutledge	70,000	6	4800
13 Emp	Smith	100,000	4	7000
14 Emp	Thorne	29,000	12	1200
15 Emp	Young	53,000	10	0

Table 10.1: Input data for 401K model

```
Employer Paid Contribution = 0.06
Desired Contribution = < Emp period data: 1 Emp - 15 Emp >
Maximum Contribution = 0.05
Rank = < Emp period data: 1 Emp - 15 Emp >
Salary = < Emp period data: 1 Emp - 15 Emp >
```

Figure 10.5: 401K variables that drive the analysis

```
Percent Difference = Highest paid percent - Lowest paid percent
. Highest paid percent = TSUM(Contribution Percentage * Highest
  paid flag) / TSUM(Highest paid flag)
. . Contribution Percentage = Total Contribution / Salary
. . . Total Contribution = Employee Contribution + Employer
      Contribution
. . . . Employee Contribution = MIN(Desired Contribution, Maximum
        Contribution * Salary)
. . . . . Desired Contribution = < Emp period data: 1 Emp - 15
          Emp >
. . . . . Maximum Contribution = 0.05
. . . . . Salary = < Emp period data: 1 Emp - 15 Emp >
. . . . Employer Contribution = Employer Paid Contribution * Salary
. . . . . Employer Paid Contribution = 0.06
. . Highest paid flag = (Rank < = Total Employees / 3)
. . . Rank = < Emp period data: 1 Emp - 15 Emp >
. . . Total Employees = COUNT(Salary)
. Lowest paid percent = TSUM(Contribution Percentage * Lowest paid
  flag)/TSUM(Lowest paid flag)
. . Lowest paid flag = NOT(Highest paid flag)
. . . Highest paid flag = (Rank < = Total Employees / 3)
```

Figure 10.6: Formulas for first 401K equity test

Emp No.	Name	Salary	Rank	Paid Flag Highest	Paid Flag Lowest
8 Emp	Levine	170,000	1	1	0
2 Emp	Dewar	155,000	2	1	0
10 Emp	O'Rourke	145,000	3	1	0
13 Emp	Smith	100,000	4	1	0
7 Emp	Jones	90,000	5	1	0
12 Emp	Rutledge	70,000	6	0	1
5 Emp	Homer	68,000	7	0	1
6 Emp	Ilkins	60,000	8	0	1
1 Emp	Bernoff	59,000	9	0	1
15 Emp	Young	53,000	10	0	1
4 Emp	Hall	36,000	11	0	1
14 Emp	Thorne	29,000	12	0	1
9 Emp	Matthews	28,000	13	0	1
11 Emp	Preston	24,000	14	0	1
3 Emp	Gallant	16,000	15	0	1

Table 10.2: Dividing employees into two groups

That is, if Rank for the current employee is less than or equal to one-third of the total number of employees, this employee is in the highest third; otherwise the person isn't.

However, you can take advantage of the fact that Javelin's comparison operators return a numeric value of 1 for true and 0 for false. It is more efficient to enter

```
Highest Paid Flag = (Rank < (Total Employees / 3))
```

This results in the value true (i.e., 1) for the rank numbers for the first third and the value false (i.e., 0) for the rest. The NOT( ) operator lets you create Lowest Paid Flag to select the other employees:

```
Lowest Paid Flag = NOT(Highest Paid Flag)
```

This turns the trues (1's) into falses (0's) and vice versa. You then compute Contribution Percentage for each employee as Total Contribution for that employee divided by Salary:

```
Contribution Percentage = Total Contribution / Salary
```

Total Contribution is the sum of Employer Contribution (6 percent of the salary) and Employee Contribution (an amount selected by the employee, subject to a 5-percent cap).

```
Total Contribution = Employer Contribution + Employee
                     Contribution
Employer Contribution = .06 * Salary
Employee Contribution = MIN(Desired Contribution, .05 * Salary)
```

Now you know what percentage of their salaries each person is contributing, and you know their rank group (highest third or lower two-thirds). What remains is to compare the average percentages of the different groups. The average percentage contributed by the highest-paid third is the sum of their individual percentages divided by the number of people in the group:

```
Highest Paid Percent = TSUM(Contribution Percentage * Highest
                       Paid Flag) / TSUM(Highest Paid Flag)
```

Breaking this formula up, the sum of the contribution percentages is

```
TSUM(Contribution Percentage * Highest Paid Flag)
```

Multiplying Contribution Percentage by Highest Paid Flag selects only the contributions of the highest third, since Highest Paid Flag is zero if they're not in that group. TSUM( ) adds up the selected percentages of all employees. To divide by the number of people in the group, you divide by TSUM(Highest Paid Flag) since Highest Paid Flag is 1 only for people in the group. Adding up all its values gives the number of people in the top-third group.

Lowest Paid Percent is computed the same way. (You could use a relative copy on a worksheet to create the formula without retyping; see Chapter 7 for how to do this.) Finally, subtract the two percentages to make the test:

Percent Difference = Highest Paid Percent - Lowest Paid Percent

Your 401K plan passes the test, because this number is less than 3 percent. So far, so good.

The second test of the plan says that the percentage contributed by the highest-paid third (Highest Paid Percent in your model) must be no more than 2.5 times that paid by the lowest-paid two-thirds (Lowest Paid Percent in your model). The following formula determines that:

Percent Ratio = Highest Paid Percent / Lowest Paid Percent

The entire model is listed in Figure 10.7, and a worksheet showing the crucial numbers is in Figure 10.8.

```
Contribution Employee % = Desired Contribution / Salary
Employer Paid Contribution = 0.06
Contribution Percentage = Total Contribution / Salary
Desired Contribution = < Emp period data: 1 Emp - 18 Emp >
Employee Contribution = MIN(Desired Contribution, Maximum
     Contribution * Salary)
Employer Contribution = Employer Paid Contribution * Salary
Highest Paid Flag = (Rank < = Total Employees / 3)
Highest Paid Percent = TSUM(Contribution Percentage * Highest
     Paid Flag) / TSUM(Highest Paid Flag)
Lowest Paid Flag = NOT(Highest Paid Flag)
Lowest Paid Percent = TSUM(Contribution Percentage * Lowest Paid
     flag) / TSUM(Lowest Paid Flag)
Maximum Contribution = 0.05
Percent Difference = Highest Paid Percent - Lowest Paid Percent
Percent Lowest Paid Count = TSUM(Lowest Paid Flag)
Percent Lowest Paid Sum = TSUM(Lowest Paid Flag * Contribution
     Percentage)
Percent Ratio = Highest Paid Percent / Lowest Paid Percent
Rank = < Emp period data: 1 Emp - 15 Emp >
Salary = < Emp period data: 1 Emp - 15 Emp >
Total Contribution = Employee Contribution + Employer Contribution
Total Employees = COUNT(Salary)
```

Figure 10.7: All formulas in 401K model

```
Total Employee      15

Contribution
 Maximum          5.000%
 Employer paid    6.000%
                First test: Compare contribution of highest
                paid third to other two-thirds.
Percent
 Highest paid    10.844%
 Lowest paid      8.462%
 Difference       2.382% <== must be no more than 3%

                Second test: Higher percent can't be more than
                2.5 times the lower percent.

 Ratio            1.281 <== must be no greater than 2.5
```

Figure 10.8: Results of 401K model

How did custom periods help you develop this model? Since the data are tied together along the time series Emp, you could use TSUM in various places to do operations to the whole series, just as for any normal time-series variable.

Checkbook Model

■ One classic application for a personal computer is to balance checkbooks. Such a model keeps track of checks and deposits, computes monthly totals and balances, and helps reconcile the checkbook with bank statements.

■ First Custom Period: CK

This model is an obvious candidate for custom time periods. Checks clearly have a time dimension (the day a check is written), but unless you are willing to be limited to writing one check per day and making one deposit per day, you can't enter check data as a daily series. Javelin's standard time periods won't do.

In this model, we subdivide months into a custom period named CK. There are as many CKs in each month as you write checks in a month, which takes advantage of the fact that different months can have different numbers of custom periods in them (as in the 5-4-4 example), with no pattern to the number of custom periods in a month. Table 10.3 shows the checks for the first three months of 1987; there are 10 checks in January, 9 checks in February, and 11 checks in March.

Check Period No.	Check Number	Check Date	Check Amount	Check Who	Check Clear
1 CK Jan 87	101	10 Jan 87	$78.97	Electric Co	1
2 CK Jan 87	102	10 Jan 87	$9.56	Gas Co	1
3 CK Jan 87	103	10 Jan 87	$69.93	Phone Co	1
4 CK Jan 87	104	10 Jan 87	$25.00	Sears	1
5 CK Jan 87	105	12 Jan 87	$72.65	Jim Smith	1
6 CK Jan 87	106	18 Jan 87	$5.71	Lechmere	1
7 CK Jan 87	107	22 Jan 87	$97.29	Exxon	1
8 CK Jan 87	108	27 Jan 87	$86.97	Mary Jones	1
9 CK Jan 87	109	27 Jan 87	$20.70	Boston Globe	1
10 CK Jan 87	110	31 Jan 87	$115.00	Rent	1
1 CK Feb 87	111	3 Feb 87	$100.06	Electric Co	1
2 CK Feb 87	112	3 Feb 87	$45.88	Gas Co	1
3 CK Feb 87	113	3 Feb 87	$53.30	Phone Co	1
4 CK Feb 87	114	10 Feb 87	$50.39	Sears	1
5 CK Feb 87	115	11 Feb 87	$39.92	Bargain Books	1
6 CK Feb 87	116	18 Feb 87	$72.40	Lechmere	1
7 CK Feb 87	117	22 Feb 87	$63.42	Exxon	1
8 CK Feb 87	118	26 Feb 87	$24.39	Safeway	1
9 CK Feb 87	119	28 Feb 87	$115.00	Rent	1
1 CK Mar 87	120	3 Mar 87	$8.26	Electric Co	1
2 CK Mar 87	121	10 Mar 87	$77.36	Gas Co	1
3 CK Mar 87	122	10 Mar 87	$49.44	Phone Co	1
4 CK Mar 87	123	10 Mar 87	$20.51	Sears	1
5 CK Mar 87	124	15 Mar 87	$53.67	Dr Grimble	0
6 CK Mar 87	125	18 Mar 87	$92.73	Lechmere	0
7 CK Mar 87	126	22 Mar 87	$14.14	Exxon	1
8 CK Mar 87	127	26 Mar 87	$21.29	Safeway	0
9 CK Mar 87	128	26 Mar 87	$5.64	Nan's Notions	1
10 CK Mar 87	129	29 Mar 87	$18.09	Kitchen Wares	0
11 CK Mar 87	130	31 Mar 87	$115.00	Rent	0

Table 10.3: Three months of checks

To begin, use / **D**efine **T**ime period to rename the A period as CK, define it in terms of months, and specify that there are 10, 9, and 11 checks in January, February, and March, respectively. Since Javelin lets you redefine custom periods when necessary, you can adjust the number of CKs in a month as you write checks.

■ *Input Variables for Checkbook Model*

On the worksheet shown in Table 10.3, Check Number, Check Amount, and Check Clear are actual variables. The Check Date and Check Who columns are just labels on the worksheet (their top borders contain the centering label-prefix ^ in front of the names so that Javelin does not try to interpret the labels as formulas). Check Number is calculated, and Check Amount is an input variable.

Check Clear is also an input variable, and is 1 for checks that have come back with a statement and 0 for those still pending. Set this variable to Fill with Blanks to make it easier to type in these values. That way, if a check clears quickly (and has its Check Clear set to 1), all previous Check Clears that haven't yet been set to 1 will be #BLANK (or 0 in Javelin 1.1). That makes writing formulas easier (no need to deal with #NA values—see Chapter 11, "Special Values and Diagnosing Problems"). On a worksheet, enter the data shown in Table 10.3, or create your own, for the input variables Check Amount and Check Clear.

One nice thing about this model is that it automatically numbers the checks for you. Check Number is defined by a formula using the time function SINCE. In the worksheet's top border, type

```
Check Number = 101 + SINCE(1 CK Jan 1987)
```

This starts the check numbers at 101 for the first CK in January 1987 and counts the number of CKs since then.

■ *Second Custom Period: Dep*

Table 10.4 shows the deposits and miscellaneous entries. Rename the B period Dep and define it in terms of months. In this case there are four, three, and four entries in the first three months. The Deposit Date and Deposit Who columns are labels, and again the border entries are preceded by ^ to

Deposit Period No.	Deposit Date	Deposit Amount	Deposit Who	Deposit Clear
1 Dep Jan 87	1 Jan 87	$512.55	Paycheck	1
2 Dep Jan 87	15 Jan 87	$44.55	Stock Dividend	1
3 Dep Jan 87	18 Jan 87	$100.00	Xmas Present	1
4 Dep Jan 87	31 Jan 87	($4.00)	Service Charge	1
1 Dep Feb 87	1 Feb 87	$512.55	Paycheck	1
2 Dep Feb 87	15 Feb 87	$44.55	Stock Dividend	1
3 Dep Feb 87	28 Feb 87	($4.00)	Service Charge	1
1 Dep Mar 87	1 Mar 87	$512.55	Paycheck	1
2 Dep Mar 87	15 Mar 87	$44.55	Stock Dividend	1
3 Dep Mar 87	31 Mar 87	$40.00	Matti Shetty	0
4 Dep Mar 87	31 Mar 87	($4.00)	Service Charge	1

Table 10.4: Three months of deposits

indicate that they are not variables. Deposit Amount and Deposit Clear are input. Deposit Clear is 1 for entries that have come back and 0 for those that have not. Enter bank service charges as deposits with negative amounts. Type in data for Deposit Amount and Deposit Clear.

■ *Calculated Variables for Checkbook Model*

Table 10.5 shows the running totals and balances. You get monthly deposit and check totals merely by showing the respective variables with monthly dates in the borders; Javelin's automatic conversion adds up the entries by month.

Let's look at the variables that give these answers. Running Total produces monthly balances. Type

```
Running Total = CUM(MONTHLY(Deposit Amount) -
                MONTHLY(Check Amount))
```

You independently convert Deposit Amount and Check Amount to monthly numbers and then subtract them. You cannot subtract Check Amount from Deposit Amount directly since one is defined in the Dep period and the other is defined in the CK period. The CUM function computes the running monthly balance. To reconcile the checkbook with the bank statement, you must separate the items that have cleared from the ones that have not. Use a trick similar to the one in the last section that separated higher- and lower-paid employees. In this case use Check Clear and Deposit Clear to mark the cleared checks, so you can define two formulas for the amounts of the cleared items:

Cleared Checks = Check Amount * Check Clear
Cleared Deposits = Deposit Amount * Deposit Clear

Then you can compute Cleared Total the same way as Running Total, to see what the bank's idea of the balance should be:

Cleared Total = CUM(MONTHLY(Cleared Deposits) –
MONTHLY(Cleared Checks))

In this case, the bank's balance is $437.41 and the balance in the checkbook is $176.63. Since the total variables represent stocks rather than flows, make sure to set their combining method on the Variable settings sheet to Last Period rather than the default; otherwise you'll get some spectacularly wrong balance numbers if you ask for quarterly or yearly balances.

Here is a trick to facilitate making entries in a checkbook like this one. Rather than updating the number of CKs and Deps in the current month every time you make an entry, set the numbers on the Custom Period settings sheet for the current month to large values like 100. Then you can type

Month	Check Amount	Deposit Amount	Running Total	Checks	Cleared Deposits	Total
Jan 1987	$581.78	$653.10	$71.32	$581.78	$653.10	$71.32
Feb 1987	$564.76	$553.10	$59.66	$564.76	$553.10	$59.66
Mar 1987	$476.13	$593.10	$176.63	$175.35	$553.10	$437.41
1 Q 1987	$1622.67	$1799.30	$176.63	$1321.89	$1759.30	$437.41

Table 10.5: Totals and balances for checkbook

in your checks and deposits just by adding new lines to the worksheets, and you still get the current balances by pressing the Calc key. This works because each input series ends some time during the current month: the number of CKs and Deps you have in the current month is just the number of them typed in so far.

At the end of each month, adjust the number of CKs and Deps for the month to the actual numbers and set the numbers for the next month to 100 on the Custom Period settings sheet. Also, display the Model settings sheet and set the default period to CK (since those, alas, are most of what you type in). Also set the end of the time span for the model to the end of this month (or later) to make sure you get check numbers for all your checks; the SINCE function only provides values out to the end of the default time span.

Figure 10.9 shows a formula listing of the whole model. This model uses many more of Javelin's time functions with custom periods. That is because your time periods are related (albeit irregularly) to the calendar. You used SINCE to count the number of periods since the beginning of the model, CUM to keep running totals, and MONTHLY to convert from custom periods to monthly data, so that you could combine them.

Defining One Custom Period in Terms of Another

■ Javelin lets you define one custom period in terms of another. For example, in the 5-4-4 production model, the first custom period is ProdWeek. If you wanted to break down your production weeks into shifts, you

```
Check Amount = < CK period data: 1 CK January 1987 - 11 CK March 1987 >
Check Clear = < CK period data: 1 CK January 1987 - 11 CK March 1987 >
Check Number = 101 + SINCE(1 CK Jan 1987)
Cleared Checks = Check Amount * Check Clear
Cleared Deposits = Deposit Amount * Deposit Clear
Cleared Total = CUM(MONTHLY(Cleared Deposits) - MONTHLY(Cleared Checks))
Deposit Amount = < Dep period data: 1 Dep January 1987 - 4 Dep March
    1987 >
Deposit Clear = < Dep period data: 1 Dep January 1987 - 4 Dep March
    1987 >
Running Total = CUM(MONTHLY(Deposit Amount) - MONTHLY(Check Amount))
```

Figure 10.9: All formulas in checkbook model

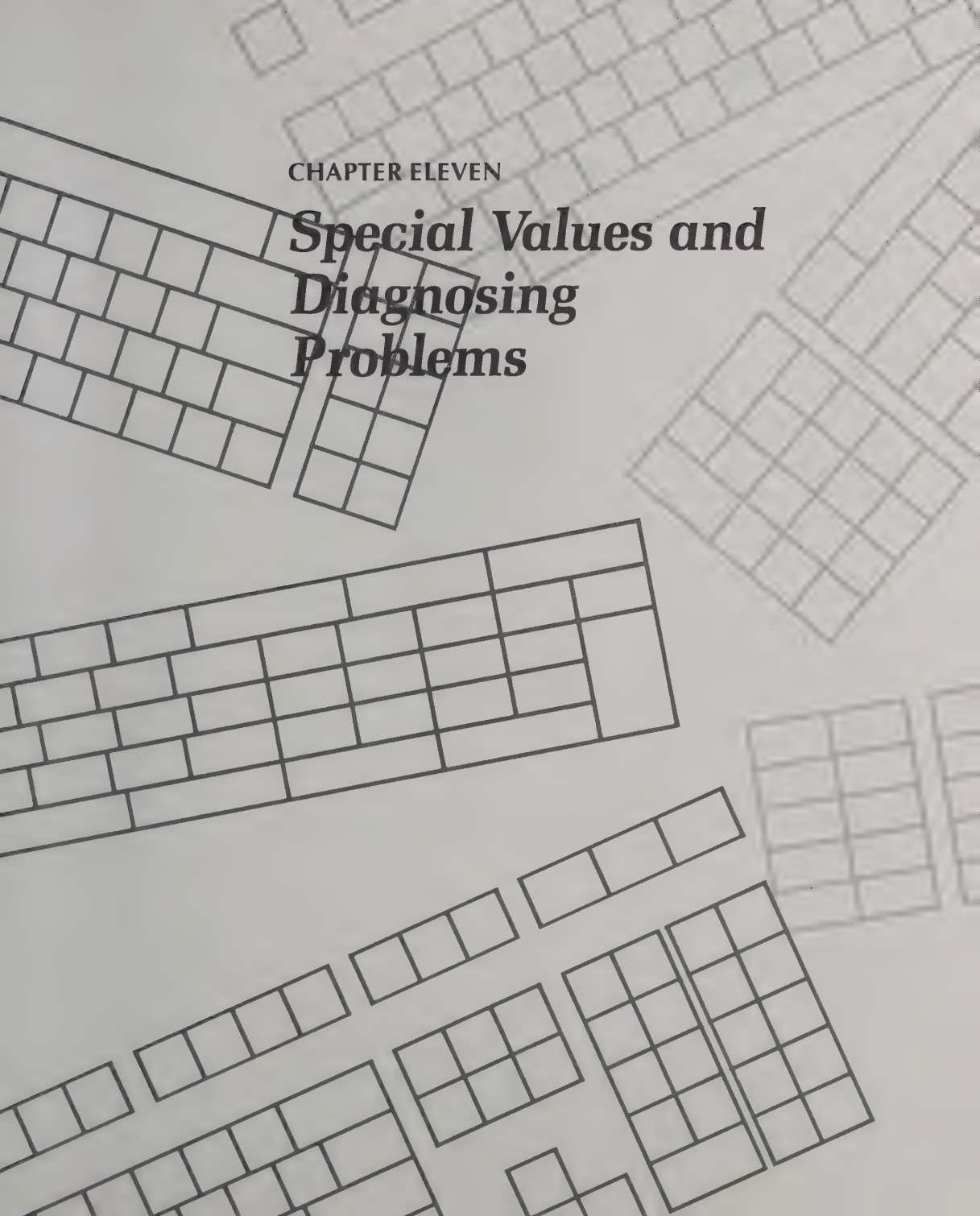
could call the second custom period Shift and define ten shifts in a production week. Both Shift and ProdWeek would be related to months.

Conclusion

■ If you have a problem that doesn't seem to "fit" in Javelin, remember to try custom periods; they are the key to using Javelin for many applications where time is not an important dimension or where your data use irregular time periods.

Even if your data don't have a periodic structure, if they have something to do with time (like the checkbook example), custom periods may be the answer. Defining your own periods is especially convenient because you can use all Javelin's time-handling features with your own time series. Don't be intimidated by custom periods; getting started with them is easy. As you saw in the 401K model, all you have to do is define the period's name to use it to group together information about anything.

We will discuss custom time periods again in Chapter 21, "Javelin Databases," when we create a database using the text and data functions that are new in Javelin PLUS. We'll also talk about how to import a custom period from one model into another in Chapter 18, "Transferring Information Between Models," so that you don't have to set up the same custom period twice.



CHAPTER ELEVEN

Special Values and Diagnosing Problems

Unlike a spreadsheet, Javelin knows a great deal about what ought to happen in your model; this is one of the advantages you get in return for the extra work of planning and building your model. As a result, when parts of your model don't match, Javelin can give you a precise indication of where the problem is, instead of producing nonsense answers as a spreadsheet might. On the other hand, this means that shortcuts that might work in a spreadsheet will not work in a Javelin model; for instance, you have to tell Javelin what to do when you want to use two series of data covering different time periods (say, Jan–June and April–Sept) in the same formula.

Javelin can identify and tell you about errors in three different places. First, when you are typing in a formula or giving a command, Javelin may be unable to complete the action you request and will show an error message on the second line of the screen. There are a great many such error messages, and in context most are quite clear. They are also listed alphabetically in the Errors chapter of the Javelin reference manual, along with some good suggestions on how to go about solving the problem, so we won't deal with them here.

A second way Javelin identifies errors is to give you a *special value* or *non-numeric value* (such as #ARITH) instead of a number as the value of a formula. You will find a list of these special values in this chapter, along with information about what causes them and suggested solutions. These special values are also listed in the Errors chapter of the Javelin manual.

Third, if Javelin cannot calculate results for a variable, it displays the variable and descriptive messages in the Errors view. You can tell that this has happened when, after you push the Calc key, you get the indicator ERROR in the lower-right corner of the screen. In this chapter, we describe how to use the Errors view with other views to help find and fix errors. Finally, you will find a discussion of why the most common Javelin errors occur, which should give you some tips on how to avoid them.

General Problems with Calculated Variables

■ There are four general kinds of problems you may have with variables: no answer at all, answers for the wrong dates, special values, and wrong answers. Let's look at each of these problems more closely.

■ *No Answers Calculated*

You may see no answer at all, such as a blank cell in the Worksheet view, CANNOT BE CALCULATED in the Table view, or a blank Chart view. The most likely causes are that you have undefined variables or that time spans of inputs do not match. See the sections "The Errors View" and "Time-Span Problems" in this chapter for suggestions on what to do.

■ *Answers for Wrong Dates*

Sometimes the range of a variable doesn't cover the period for which you want answers. The spans of inputs may overlap only for the periods for which you are getting answers, rather than those for which you want answers. BEGIN, END, or RANGE functions may not be set up correctly, or your default time span may not be right. See the section "Time-Span Problems" for suggestions.

■ *Special Values*

Javelin displays special values or non-numeric values when it cannot calculate an answer for one of a number of reasons. These values appear as #NA, #NODATE, #ARITH, and others. Each special value indicates a different kind of problem. Those errors and possible solutions are described in "Special Values" below.

■ *Wrong Answers*

Don't believe your model's results until you have checked them to make sure they are reasonable. It is easy to forget to convert interest rates from annual to monthly and to make other logic errors. Most complicated models, even ones that seem to be working correctly, still have a few bugs.

Sometimes it appears that you have entered your formula correctly but Javelin computes the wrong answer anyway. Although it is possible that Javelin itself has a bug, it is much more likely that the model contains subtle errors you have not found yet. We used Javelin for over a year while writing this book and found only one Javelin bug, although we found a lot of errors in our models.

Remember, as with any tool, you are responsible for how well you use Javelin. If your model does not give you the answer you expect, check it carefully, try creating simple cases of areas where you are having problems, and finally, try to build up to your problems step-by-step before you conclude that you have found a Javelin problem.

Throughout this book we have included tips that should help you avoid the most common problems. The most important factor is to make your model say as accurately as possible what you mean it to say. For example, to use a 15-percent annual interest rate in a monthly calculation, say

```
Interest Rate = .15
Interest Expense = Loan Balance * MONTHLY(YEARLY(Interest
                   Rate))
```

MONTHLY(YEARLY( )) tells Javelin that you want to convert the yearly rate to a monthly one.

You could have written the same formula in one of the following ways, but if you change Loan Balance to some period other than monthly, you'll get the wrong answer with no notice that there was a problem.

```
Interest Expense = Loan Balance * .0125
Interest Expense = Loan Balance * (.15 / 12)
Interest Expense = Loan Balance * Interest Rate / 12
```

Diagnostic Tools

■ Javelin provides several powerful tools to help you check your models for errors and to fix those errors. The Errors view shows the variables in which Javelin has found problems, and the Roots and Branches views let you trace the sources of your results and the effects of your changes.

■ The Errors View

The Errors view shows you a list of the variables in your model that have an error or are undefined. After each formula, there is a message describing the nature of the error. When you are in the Errors view, you can use

commands in the / *Errors menu to limit the kind of errors shown. Table 11.1 contains a list of the commands. There are six error messages that may appear after the formula for the variable, described in Table 11.2.

■ The Roots and Branches Views

The Roots view is similar to the Formulas view, except that rather than displaying all the variables in the model, it starts with a single variable and then shows the other variables on which that variable depends. Roots is useful because, once you know what variable is in error, Roots directly tells you how that problem variable is computed. In a complicated model, a variable may depend on dozens of other variables; you can use the commands on the Roots star menu to limit the display to a manageable size (see Table 11.3).

When tracing through the logic of a model, you can put the Roots view in one window, the Table view in the other, and then sync the two windows together. As you use the cursor keys to move through the Roots view, you can see the data for each variable, making it easier to track down particular problem values or to see where particular results come from. You will see the Roots view used to diagnose practically all the errors encountered in this chapter.

Time-Span Problems

■ Most difficulties with empty output variables or wrong dates are caused by time-span problems. Here are the most common ones.

Command	Description
All errors	Display all variables that have errors. This is the same list that you see when you first display the Errors view.
Undefined	List only variables that are undefined.
Time period	List only variables that have time period problems, such as mismatched time periods in a formula.

Table 11.1: The / *Errors commands

■ *Mismatched Input Time Spans*

One frequent problem is to have a calculated variable that seems to calculate correctly but ends up with no data. Most often, the time spans for the variables upon which the problem variable depends do not overlap at all. Javelin PLUS (when not in 1.1 compatibility mode) changed the way time spans are determined to alleviate this problem, but it still occurs sometimes.

The Errors and Roots views help track down this kind of problem. In Figure 11.1 you see the Errors and Roots views synced together. Variable C depends on defined variables A and B but has no result values. You use

Message	Explanation
UNDEFINED	Undefined variable. The variable has been defined neither as an input variable, by entering some values, nor as a calculated variable, by entering a formula.
Formula uses mixed or invalid periods	The formula combines daily, weekly, monthly, quarterly, yearly, and/or custom periods without using the necessary conversion function. To combine values of different period, you must use a conversion function such as MONTHLY() to convert the period of one to match the period of the other (see Chapter 9).
Time-span error	No dates can be computed for formula. The time spans of the variables mentioned in the formula do not overlap, or a time span function such as RANGE() cannot come up with a useful time span. See the section "Time-Span Problems."
Argument (*n*) to (*function*) is constant, should be time series	Javelin functions require certain kinds of arguments. See Chapter 8 for which function arguments must be constants and which must be time series.
Argument (*n*) to (*function*) is time series, should be constant	Again see Chapter 8 for which function arguments must be constants and which must be time series.
Out of memory	Javelin couldn't recalculate this formula because it ran out of memory.

Table 11.2: Messages in the Errors view

/ **V**iew **E**rrors in the top window and / **V**iew **F**ormulas **R**oots in the bottom window, then push F3, the Sync key, in the top window to make the Roots view show the roots of the error variable. In this case you can look at the Roots view and see the problem pretty quickly: variable A runs from January through March, but B runs from April through June, so they don't overlap.

Now let's look at a slightly more complex and realistic example. In Figure 11.2 is a piece of a model that computes Gross Profit from Revenues Total and Expenses Total, each of which is computed from individual state

Command	Description
All roots	Display all roots of a variable. In Javelin 1.1, this shows every variable every time it occurs in a formula. In Javelin PLUS, no variable is shown more than once.
Inputs	Show only roots that are input variables, not calculated ones.
Undefined	Show only roots that are undefined. Convenient for tracing CANNOT BE CALCULATED errors.
Notes	Show only roots that have notes.
Time spans	Show the time spans of roots rather than the formulas. Convenient for tracking down problems due to mismatched time spans.

Table 11.3: The / *Roots commands

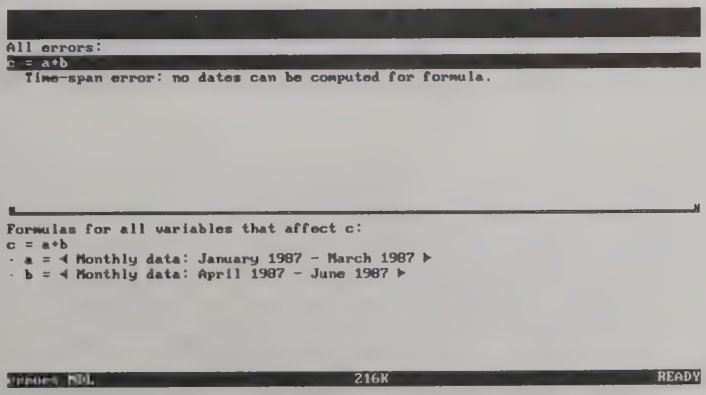

All errors:
c = a+b
Time-span error: no dates can be computed for formula.
Formulas for all variables that affect c:
c = a+b
· a = ◂ Monthly data: January 1987 - March 1987 ▸
· b = ◂ Monthly data: April 1987 - June 1987 ▸
216K
READY

Figure 11.1: Tracking down a time-span problem with the Roots view

numbers. Again we've synced the Errors and Roots views. If you look at the Roots view, you can probably find the problem, but the number of formulas is rapidly reaching the point where it is too hard to figure out the time spans of everything in your head.

Fortunately, Javelin is willing to tell you the time spans that it is using, so there is no need to figure them out yourself. Use / * Roots **T**ime **S**pans in the lower window to see the time spans of all of the variables involved, as in Figure 11.3.

Here, Gross Profit is "Monthly calculated," which means that Javelin can tell that it is Monthly but cannot compute any dates, which you already knew. The problem must be in the Revenue or Expenses variables. If you look at the variables at the next level, you see that Total Revenue runs from April through June, but Total Expenses runs from January through March—here are variables that don't overlap. A little more investigation shows that the problem is in New Hampshire Revenue and New Hampshire Expenses, probably because you forgot to type in all the information.

■ *Other Time-Span Problems*

Although lack of input data is the most common cause of time-span problems, there are a few other possibilities to be aware of. For example, if you type

Campaign Overhead = RANGE(Jul 1987, Jan 1987, 15000)

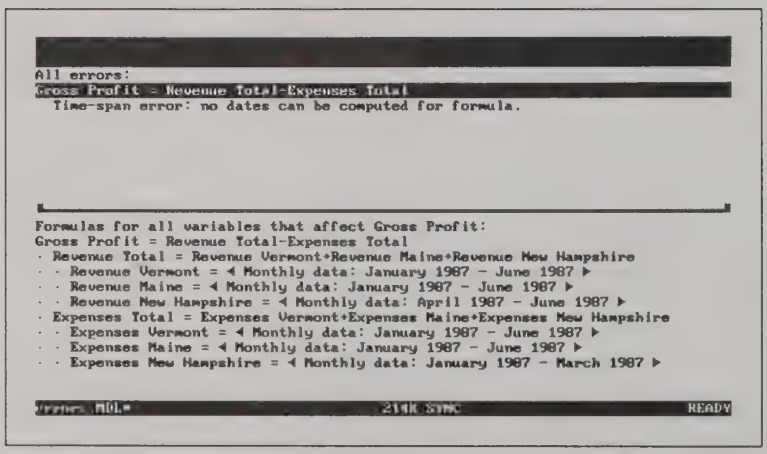

All errors:
Gross Profit = Revenue Total-Expenses Total
Time-span error: no dates can be computed for formula.
Formulas for all variables that affect Gross Profit:
Gross Profit = Revenue Total-Expenses Total
· Revenue Total = Revenue Vermont+Revenue Maine+Revenue New Hampshire
· · Revenue Vermont = ◀ Monthly data: January 1987 - June 1987 ▶
· · Revenue Maine = ◀ Monthly data: January 1987 - June 1987 ▶
· · Revenue New Hampshire = ◀ Monthly data: April 1987 - June 1987 ▶
· Expenses Total = Expenses Vermont+Expenses Maine+Expenses New Hampshire
· · Expenses Vermont = ◀ Monthly data: January 1987 - June 1987 ▶
· · Expenses Maine = ◀ Monthly data: January 1987 - June 1987 ▶
· · Expenses New Hampshire = ◀ Monthly data: January 1987 - March 1987 ▶
214K SYNC
READY

Figure 11.2: More complex time-span problem

Campaign Overhead will show up calculated as Monthly with no data, because the two dates are in the wrong order. This kind of problem can be particularly troublesome in Javelin PLUS, since the dates used by RANGE( ) can themselves be calculated. Syncing the Table view to the Roots view usually lets you find the problem data.

You can also get into trouble if you don't pay attention to your default date range. Assume your default date range runs from January 1987 through December 1987, and you have this formula:

```
Product Acceptance = GROW(1 @ Dec 87, .01)
```

If you look at Product Acceptance in the Table view, you'll find that it doesn't grow at all. It's just the single value 1 in December 1987. The reason for this is that GROW( ) runs out only to the end of the default date range and no farther.

You might be tempted to fix the problem by using a RANGE( ) function to force the range like this:

```
Product Acceptance = RANGE(Dec 87, Dec 88, GROW(1 @ Dec 87,
                     .01))
```

That doesn't work, because RANGE( ) can only restrict the range of an expression, not make it longer. You can have the same problem using the ; operator:

```
Revenue = Sales; PREVIOUS(Revenue) * 1.01
```

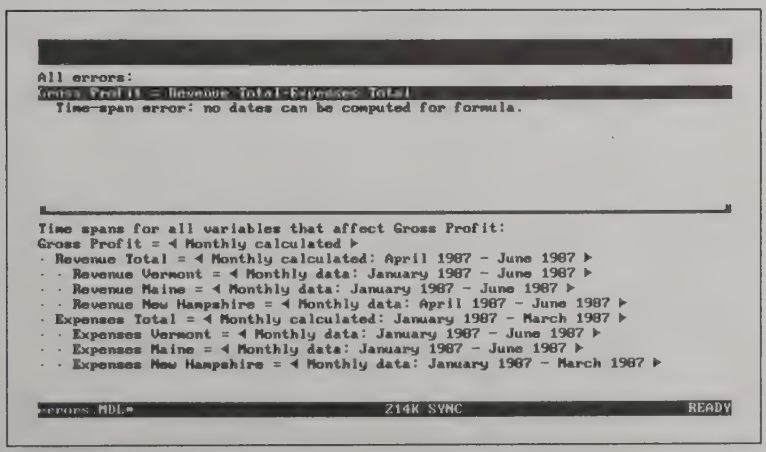

All errors:
Gross Profit = Revenue Total-Expenses Total
Time-span error: no dates can be computed for formula.
Time spans for all variables that affect Gross Profit:
Gross Profit = ◄ Monthly calculated ►
· Revenue Total = ◄ Monthly calculated: April 1987 - June 1987 ►
· · Revenue Vermont = ◄ Monthly data: January 1987 - June 1987 ►
· · Revenue Maine = ◄ Monthly data: January 1987 - June 1987 ►
· · Revenue New Hampshire = ◄ Monthly data: April 1987 - June 1987 ►
· Expenses Total = ◄ Monthly calculated: January 1987 - March 1987 ►
· · Expenses Vermont = ◄ Monthly data: January 1987 - June 1987 ►
· · Expenses Maine = ◄ Monthly data: January 1987 - June 1987 ►
· · Expenses New Hampshire = ◄ Monthly data: January 1987 - March 1987 ►
Z14K SYNC
READY

Figure 11.3: Using / *Roots Time Spans to see time spans

Revenue will have data only to the end of your default date range. The ; operator does not return values outside the default date range when used in a spiral variable like this one.

The way to fix these kinds of problems is to make the default date range longer, so that it includes the entire time span that is of interest in the model. When in doubt, make the default date range longer than the date range that you are interested in. Once your model works, cut the date range back; your model will be smaller and calculate faster if you do.

■ *Fixing Time-Span Problems in Javelin 1.1*

Sometimes time-span problems are caused by forgetting to put in part of your data. More often they are due either to the way that Javelin determines the time spans of calculated variables when you do arithmetic or else to problems getting spiral models started.

In Javelin 1.1 (or in compatibility mode), when you tell Javelin to combine several variables with addition or arithmetic functions such as AVG, the time span of the result is the *intersection* of that of the inputs. That is, it computes a result only for the period when all the inputs are defined. Type

```
Total Sales = Regular Sales + Promotion Sales
```

In this case, assume that Regular Sales runs from January through December, but the promotion was only in the summer, so Promotion Sales runs from June through August. Javelin would come up with an answer only for June through August—hardly what you want. The semicolon operator is the best way to handle this:

```
Total Sales = Regular Sales + (Promotion Sales; 0)
```

This says that where Promotion Sales is undefined, use a value of zero. The parentheses are important here. Had you written

```
Total Sales = Regular Sales + Promotion Sales; 0
```

Javelin would interpret it as

```
Total Sales = (Regular Sales + Promotion Sales); 0
```

and it would return zero as the answer except for June through August.

Except in the simplest formulas, you should almost always put parentheses around semicolon expressions.

■ Time Spans in Javelin PLUS

Javelin PLUS computes time spans in a very different way than Javelin 1.1. Rather than computing values for the intersection of the time spans of the variables in a calculation, Javelin PLUS calculates values for all dates in the *union* of the time spans. That means that in Javelin PLUS, all the examples in the previous section would work without the semicolon operator.

More precisely, here is what Javelin PLUS does (when not in compatibility mode). For each variable, all dates outside the range of the variable are considered to be filled with #BLANK, the new name for #RANGE. In addition and subtraction with numbers, #BLANK acts like a zero, so that adding or subtracting #BLANK to or from a number gives that number. For all other operations, #BLANK behaves like the other special values, so that multiplying or dividing by #BLANK gives #BLANK. In most cases, this gives the desired result, so that if you are missing some of your data, you can still add up the rest. Functions that skipped #RANGE values, such as TAVG( ) and RMIN( ) still skip #BLANK, so you can still do things like compute the average of a set of data and not count the missing points.

■ Time Span Problems in Spiral Models

The semicolon operator is also useful to get spiral models started. Let's say that in your cash flow model you wrote this:

```
Balance = PREVIOUS(Balance) + Deposits - Withdrawals
```

In Javelin 1.1, you'd find that all the values for Balance would calculate as #RANGE. Why? Javelin starts to calculate the Balance value for the first period, so it gets the first value for Deposits and Withdrawals, and looks for the previous value of Balance. But this is the first period, so there is no previous Balance, and Javelin returns a #RANGE. Then each subsequent Balance value fetches the #RANGE from the previous date, and you end up with all #RANGEs. This will calculate the correct answer:

```
Balance = PREVIOUS(Balance; 0) + Deposits - Withdrawals
```

When Javelin looks for the previous balance for the first period, now ;0 tells it to use zero if there is no Balance. This lets it calculate the first period, then all the rest of the periods are calculated as normal.

When doing a running sum with PREVIOUS, you could probably do it better with CUM, so you could rewrite this formula as

```
Balance = CUM(Deposits - Withdrawals)
```

Deposits and Withdrawals might not both run for the entire time span, so in Javelin 1.1 you might even need this:

```
Balance = CUM((Deposits; 0) - (Withdrawals; 0))
```

Circular References

■ A *circular reference* is a formula that depends on itself, or a series of formulas that refer to each other, for example,

```
Balance = Balance + Deposits - Withdrawals
```

When you have a circularity like this one, you probably intend your model to be spiral rather than circular and usually you have forgotten to use the PREVIOUS function. This formula can be corrected by basing this period's Balance on the previous period's Balance:

```
Balance = PREVIOUS(Balance) + Deposits - Withdrawals
```

In this example, you'll probably have to do something to get the Balance series started, as discussed in the previous section. More often, though, several formulas will form a circular reference:

```
Profit = Revenues - Expenses - Overhead - Bonus
Bonus = 0.2 * Adjusted Profit
Adjusted Profit = Profit - Extraordinary Items
```

In cases like this, you want a set of formulas to be circular. By repeated calculation, you can usually get closer and closer to the correct answer until

the model *converges* to the correct number. Here, you need to give Javelin a number to start with for one of the values in the circle so that it can get started calculating the circular set of formulas. This number to start with is called the *seed*. The choice of seed depends on the particular problem, but you want to make it as close as possible to the eventual answer. If you don't have an educated guess to use, zero is usually a reasonable figure to start with.

To seed a set of circular references, enter the number you want to start with in the Table or Worksheet view replacing the #CIRC indicator. This is *the only place* Javelin lets you type in a value for a variable that is defined by a formula. Everywhere else, you will be given an error telling you that the variable is defined by a formula. In this case, because the existing value is #CIRC, Javelin knows that you are typing in a seed value that will be replaced when the calculations complete.

Then press the Calc key repeatedly and watch the values of the circular formulas change. They should converge on an answer, but not every set of circular formulas will converge. If the values of your circular formulas do not start to converge after a few pushes of F9, your model may not converge and you must reformulate it to remove the circular references.

■ *A Compensating Balances Model*

Frequently, when a business takes out a loan from a bank, the bank requires that the business leave some fraction of the loan on deposit. (This is an indirect way of raising the effective interest rate.) Figure 11.4 shows a simple model that computes the monthly amounts of loans needed, assuming that you have to leave 10 percent on deposit as a compensating balance.

To make the circularity clearer, you can look at a Roots view of Total Loan, as in Figure 11.5. Javelin automatically notices when a variable depends on itself and shows it in the Roots view as CIRCULAR DEFINITION.

```
Formulas for all variables:

Cash Loan = 15000
Compensating Balance = 0.1 * Total Loan
Total Loan = Cash Loan + Compensating Balance
```

Figure 11.4: First Compensating Balances model

Use / *Formulas Circular to see the formulas involved in the circular reference. Then view one of the circular variables in the Table or Worksheet view and type seed values over the #CIRC entries. In this case, a good variable to seed is Compensating Balances. If you seed it with zero, it will converge to within a dollar after eight presses of F9 and converge exactly after eleven presses.

If you subsequently change the inputs to your circular model, each time you press F9 it will move from the previous answer toward the correct answer for the new inputs. If the new inputs are very different from the old ones, you may need to reseed the circular variables. You can write Javelin macros to automate the job of seeding a circular model and then recalculating it until it converges (see Chapter 14, "Macros").

Special Values

■ Javelin has ten special values that it uses instead of numbers to show that for some reason there is no numeric value for a variable. These values usually appear preceded by the sign #. In the Table view, they may be replaced by the message NOT YET CALCULATED, which means that the variable contains #CALC, or CANNOT BE CALCULATED, which means that a calculated variable contains #UNDEF. In addition, if any formula tries to use a variable whose value is a special value, that formula's value is also the special value.

For example, let's say that you decide to start a small company, put a few ads in the paper, and want to compute your advertising cost per sale. Figure 11.6 shows the formulas. Ad Expense and Cost per Sale are inputs, and you compute Cost per Sale by dividing one by the other. Figure 11.7 contains the answers, which are fine except that the May value for Cost per Sale is #ARITH, since you sold no units that month and divided by zero.

```
Formulas for all variables that affect Total Loan:

Total Loan = Cash Loan + Compensating Balance
. Cash Loan = 15000
. Compensating Balance = 0.1 * Total Loan
. . Total Loan = < CIRCULAR DEFINITION >
```

Figure 11.5: Roots view of first Compensating Balances model

There are many ways to deal with special values. In this case, we have adjusted the logic to deal with the problem: Adjusted Cost per Sale uses an IF( ) function to deal with the case in which Units Sold is zero.

Each special value means something different, so we'll consider individually where they come from and how to overcome them.

■ #ARITH: Arithmetic Error

#ARITH means that there was, arithmetically, no answer to your formula. For example, dividing by zero, SQRT(−1), and LOG(−1) does not produce real answers. Usually this means that your arithmetic is wrong or that something is happening that you hadn't anticipated, such as the case mentioned above of zero units sold in a month. You can use the Roots and Table views to check that your inputs are correct and to see what values are causing #ARITH. Then you can either change the inputs or change the logic as shown above.

The IRR( ) function sometimes cannot compute an answer and returns #ARITH, either because no answer exists for the values input or because the method it uses to search for the answer by refining consecutive guesses could not find it. The former case usually occurs when all input values are positive or all are negative, since IRR( ) requires inputs of both signs to compute an answer. The latter case usually occurs when the answer is not close

```
Ad Expense = < Monthly data: March 1987 - June 1987 >
Adjusted Cost per Sale = IF(Units Sold = 0, 0, Ad Expense / Units Sold)
Cost per Sale = Ad Expense / Units Sold
Units Sold = < Monthly data: March 1987 - June 1987 >
```

Figure 11.6: Model with divisions by zero

	Mar 1987	Apr 1987	May 1987	Jun 1987
Ad Expense	1000.00	1100.00	1200.00	1100.00
Units Sold	3	2	0	3
Cost per Sale	333.33	550.00	#ARITH	366.67
Adjusted Cost per Sale	333.33	550.00	0.00	366.67

Figure 11.7: Inputs and results of the model

to Javelin's first guess of 10 percent. You can give IRR( ) a second argument that tells it what guess to start with. You can usually get an answer by trying various guesses based on your estimate of the answer.

■ #CALC: Not Yet Calculated

When you create or change a variable defined by a formula, Javelin sets its value to #CALC until you press F9, the Calc key, to recalculate. If you interrupt a long recalculation with Ctrl-Break, the values that remain to be calculated will contain #CALC.

■ #CIRC: Circular Reference

A value is defined directly or indirectly in terms of itself. See the section "Circular References" above.

■ #ERROR: Formula Too Complicated

A formula is too complicated for Javelin to evaluate. Your formula has to be enormously complicated to cause this error; we have never seen one that did. Should you run into this value, break your formula into two.

■ #INF: Infinity

A number is too large for Javelin to handle. You must change the formula or input data to fix this error. Javelin can handle up to three-hundred-digit numbers, so this error usually indicates a logical error in a model. The most common is taking an EXP or power of large numbers incorrectly. Again, the Roots and Table views offer a good way to find the values and formulas that are causing trouble.

■ #NA: Not Available

This signifies that no value for the variable exists at that date. If you leave a gap between values in an input variable in Javelin 1.1, the gaps will be filled with #NAs. Any other variable that depends on that input variable will also

get #NA values wherever the missing period's values are used. If appropriate, you can fill in the missing values in the input variable.

In Javelin 1.1, you can also use the Variable settings sheet to change the setting to Fill with 0's if appropriate, in which case all #NAs will change to zeros. Once you have changed the #NAs to zeros, there's no automatic way to change them back. In some cases, you can change the logic of the model not to depend on the missing data, either by using IF( ) statements or by using functions such as TSUM( ) and RAVG( ), which skip over #NA values.

In Javelin PLUS, rather than filling empty points with zeros, Javelin fills them with #BLANK. Since #BLANK behaves like a zero when you add it or subtract it to or from a number, the effect of filling with #BLANK is similar to the effect of filling with zeros.

You can still tell Javelin to fill missing points in a variable with #NA, which is the right thing to do when you want to be sure not to try to calculate answers that depend on the missing data. In the more common case that you can safely skip the missing data, filling with #BLANK is the better choice.

Figure 11.8 shows another way to deal with #NAs. Raw Houses Sold is a series that contains #NA values. Houses Sold is a calculated series that uses an IF( ) function to substitute zeros for the #NAs. You can use this trick to deal with most special values, except #CALC and #CIRC, which have special meanings within Javelin's recalculation process.

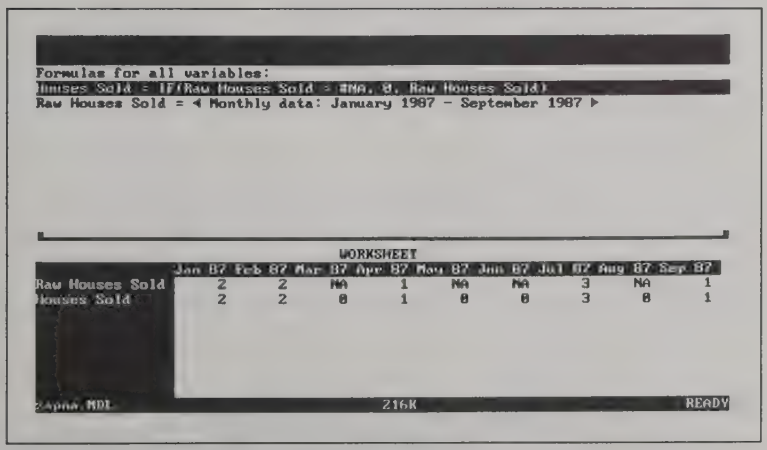

Formulas for all variables:
Houses Sold = IF(Raw Houses Sold = #NA, 0, Raw Houses Sold)
Raw Houses Sold = ◄ Monthly data: January 1987 - September 1987 ►
WORKSHEET
Jan 87 Feb 87 Mar 87 Apr 87 May 87 Jun 87 Jul 87 Aug 87 Sep 87
Raw Houses Sold 2 2 NA 1 NA NA 3 NA 1
Houses Sold 2 2 0 1 0 0 3 0 1
216K
READY

Figure 11.8: Calculating around #NA values

■ #NODATE: Date Needed for Variable

This special value occurs mostly in the Worksheet view, when you have used the border cells to specify the name of the variable but not the date. For example, if you have a time series Eastern Sales, and put Eastern on the top and Sales on the side, you get this error because Javelin cannot tell what date you want. To display a value, you have to put a date into one of the borders to tell Javelin what date to use.

You'll also get #NODATEs if you define a custom time period and then do some kinds of date conversions. If on a worksheet you try to display a variable defined in a noncalendar custom time period using calendar dates you will see #NODATE. This is an error because no conversion is possible unless you define the custom period in terms of a calendar period.

Even custom time that is linked to the calendar can cause #NODATE. Say that you define the time period A in terms of months, and there are three A's in each month from January to May 1987. Then, if you try to convert a series defined in A's to yearly, you get #NODATE because there are not dates for the whole year. The solution here, as you might expect, is to go into the Custom Period settings sheet and define the A's to the end of the year.

■ #OUTMEM: Out of Memory

Your PC does not have enough memory for Javelin to complete the recalculation of your model. The variables that it didn't get to before running out of memory have #OUTMEM values. You can split the model up into smaller models, each of which will fit. See Chapter 18, "Transferring Information Between Models," for how to do this. You might be able to shrink the default date range or recast some daily calculations as monthly or quarterly to decrease the amount of storage needed. Ten years of daily data take almost 30K to store, while ten years of months take about 1K. Alternatively, you can remove memory-resident programs that you have installed or add expanded memory to your computer to increase the size of model it can handle.

■ #RANGE: Out of Time Range (Javelin 1.1)

You are using a date that is out of the time span for which the variable is defined. See the section "Time-Span Problems" above. You can usually use the semicolon operator to fix your formulas.

■ #BLANK: No Value Assigned (Javelin PLUS)

Javelin PLUS has renamed #RANGE to #BLANK, and has changed its meaning. In Javelin 1.1 (or in compatibility mode), adding or subtracting a number and a #RANGE gives #RANGE, while in Javelin PLUS a #BLANK behaves like a zero. For any other operations, such as multiplication or TAVG( ), #BLANK still acts as #RANGE did, so that TAVG( ) and similar functions disregard it, and multiplying by #BLANK gives #BLANK.

In Javelin 1.1, models needed complex formulas to permit columns of numbers to be added up when some of the values were missing. #BLANK is now a placeholder that permits these calculations to proceed without the extra complexity.

■ #UNDEF: Undefined Variable

Variables are defined either by entering values for them or by entering a formula that defines their values. As the name implies, an undefined variable has not been specified either way. Undefined variables appear if a formula refers to a variable that has *not yet* been defined, presumably because you're not through typing in your model.

They can appear also if you clear a variable that is used in another formula. When you clear the variable, its formula and data disappear, but the reference to it remains. You must rewrite or delete the formula that refers to the undefined variable to make it disappear. Another easy way for undefined variables to appear is through typing errors; remember that variable Widget Cost is different from variable Widgets Cost and both are different from variable Wigdet Cost.

The Branches view is the best way to find out where your undefined variables are used. If you display an undefined variable in the Branches view, Javelin will show you directly the formulas that depend on it. If you sync the Branches view to the Errors view and use / *Errors Undefined to see the undefined variables, you can find all references to undefined variables by moving the cursor through the Errors view.

A variable could also give the value #UNDEF because it depends on a variable that is undefined. In this case, you either have to find and define the undefined variable or else rewrite your formula not to depend on undefined variables. You can find undefined variables with the Errors view, or use the Roots view to see the variables that your formula depends on and use / *Roots Undefined to show just the undefined roots.

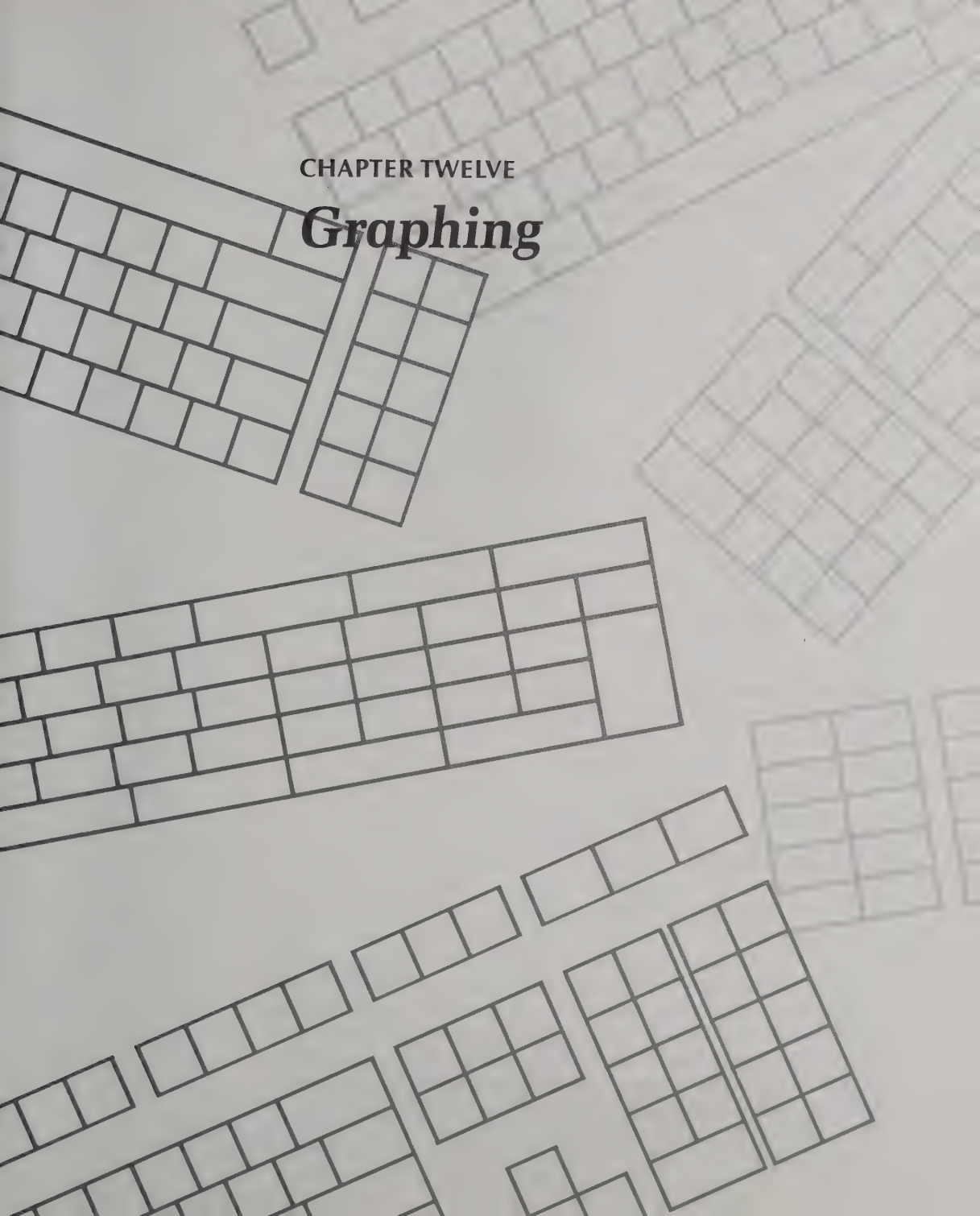
Conclusion

■ A Javelin model has much more internal structure than an equivalent spreadsheet. Therefore, Javelin can spot errors that could generate nonsensical results, and it can give a lot more help in determining what is wrong with a model than a spreadsheet program can. Errors in Javelin models manifest themselves in three places: on the error line (if they are detectable when typed); in the special values that may appear instead of answers in a variable; and in the messages of the Errors view.

The Errors view is useful for tracing the causes of errors, usually in connection with the Roots view. In particular, as you scroll through the Errors view with the windows synced, the roots of the formula generating the error will appear. Special values help you diagnose individual formulas, most often pointing to an improper use of time or improper arguments to a function.

Finally, we have reviewed a few useful techniques to make your model more robust, like circular and spiral models, making your time conversions for constants explicit, grouping variables and semicolons together with parentheses, and using special values in formulas. All these techniques give you a better chance of building a model that will work not only when you first develop it but much later when you come back to use it with different data, or when you pass it along to a colleague.

Javelin helps make sure that your answers are correct by requiring you to tell it explicitly what to do when data are missing or when you want to combine data from different periods. But you should always have a ballpark idea of what the answer to your model should be, to check up on both your model and Javelin.



CHAPTER TWELVE

Graphing

There are three views that display information graphically in Javelin: Quick-graph, Chart, and Graph. In Chapter 6, we introduced the Quick-graph and Chart views, which display graphs of one variable at a time and allow you to enter and edit values for the variable. Now, let's look at the Graph view, which lets you create presentation graphics.

Javelin can create seven different kinds of graphs (eight in Javelin PLUS), most of which can show up to seven variables at once. In addition, there are options for titles, scaling, legends, and other features. However, you cannot change the values for the variables in the Graph view—the purpose of this view is to create and print graphs for analyzing and presenting information in a model.

In this chapter, you will create, display, edit, and print Javelin graphs using the Graph view. You'll use the commands / **D**efine **G**raph, / **V**iew **G**raph, and / **P**rint **G**raph, and the Graph key, F10.

The Sample Model: ECON

■ In this chapter, you'll use the ECON model that you created in Chapter 8. It contains data about the U.S. economy, extracted from the *Economic Report of the President,* February 1986. Load the model to be ready to define some graphs.

Hardware Requirements

■ To display graphs, you must have a screen and a monitor adapter board that can display graphics. Javelin can display graphics on a number of monitor adapter boards, including the IBM Color/Graphics (CGA) board, the IBM Enhanced Graphics Adapter (EGA) board, the Compaq combined text and graphics screen, the Hercules video board, and other boards compatible with them. The most common monitor adapter on which Javelin graphs do not work is the IBM Monochrome Adapter, which cannot display graphics at all. (Javelin displays an error message if you try to display a graph on a system without the necessary hardware.)

To print graphs, your printer must be able to print graphics. Dot-matrix, ink-jet, and laser printers generally can print graphics, while daisy-wheel printers cannot. Javelin also supports several Hewlett-Packard (HP) plotters and other plotters compatible with them.

When you install Javelin, it shows a list of screens and printers that it works with, and you must specify the one you have. In addition, you can select printers and screens by name by using the commands / **D**efine **P**rinter and / **D**efine **C**onfiguration.

Using Graphs

■ Graphs are like worksheets in that they give views of your data and have names, and you can have more than one of them in your model. To create a graph, use the command / **D**efine **G**raph. This command displays the Graph settings sheet where you choose the variables to be graphed and options about how the graph will appear. To display the graph, use the command / **V**iew **G**raph or F10, the Graph key. To print the graph, give the command / **P**rint **G**raph.

Since you can make as many graphs as you like in a model, you must give each one a name when you create it. To tell Javelin which graph you want to display or print, indicate the graph name in the / **V**iew **G**raph or / **P**rint **G**raph command. Like worksheets, macros, and other views, graphs are stored as part of the Javelin model when you save it on the disk.

When you display a graph, it takes up the entire screen. The view(s) on the screen are temporarily covered up by the graph. When you press Esc, the graph is removed from the screen and the same view(s) are displayed again.

Javelin's eight different types of graphs are listed in Table 12.1.

Creating a Graph

■ To create a new graph, you use / **D**efine **G**raph. Javelin asks you to give the graph a name, and then displays the Graph settings sheet. You can enter information about what variables to graph and how the graph should look.

Give the command / **D**efine **G**raph, then name this graph. Type

GNP Current Dollars

Graph names can be over a hundred characters long and can contain letters, numbers, and spaces, but no punctuation. Now you see the Graph settings sheet (see Figure 12.1).

Type	Description	Effect
Line	Shows each value for each of up to seven variables as a point, with the points connected by a line. Variables are distinguished by different symbols at the points or by different types of lines connecting the points.	Emphasizes continuity of each variable, rather than comparison between values for a given period.
Bar	Shows each value for each of up to seven variables as a vertical bar. The bars for the variables are set side-by-side and distinguished by color or shading.	Emphasizes comparison between values for given period, rather than continuity of each variable.
Stacked bar	Shows each value for each of up to seven variables as a vertical bar. The bars for the variables are stacked one above the other, so the total height of the bar shows the sum of the variables.	Useful when total of the variables is also a meaningful value.
Percent bar	Shows each value for each of up to seven variables as a vertical bar, with the bars for the variables stacked up. However, the total height of the bars is constant for all periods, and the heights for the individual variables are shown as percentages of the total bar height.	Useful when total of the variables is a meaningful value and you want to convey how the variables combine to make the total.
Horizontal bar	Shows values for each variable in one period as a horizontal bar.	Useful to show values for one period.

Table 12.1: The Javelin graph types

Pie chart	Shows each value for one variable as a wedge in a circle, where the size of the wedge is proportional to its part of the total of the values.	Useful to show how values for one period combine to make up their total.
XY Scatter plot	Using one variable for x values plots up to six variables as y values against that x variable (Javelin 1.1 allows only one y variable.)	Does not show the time for each point, just the way the two variables interact.
High-Low-Close	Shows high, low, or closing stock or commodity prices as floating vertical lines	Useful for stocks and commodities. Also allows one bar to be combined with up to three lines.

Table 12.1: The Javelin graph types (continued)

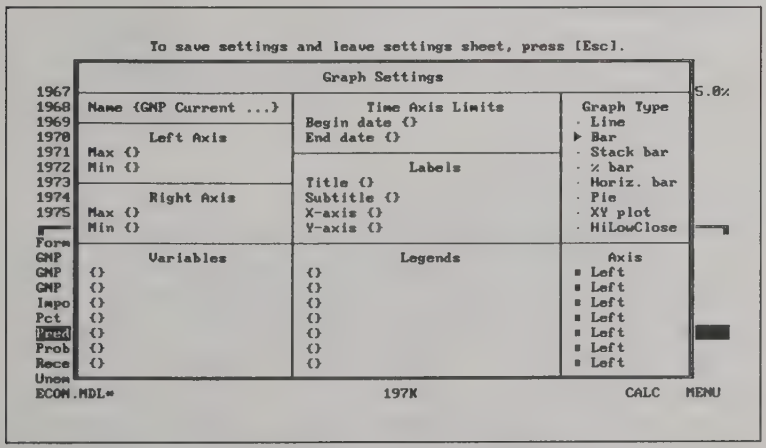


Figure 12.1: The Bar and Line Charts Graph settings sheet

■ *The Graph Settings Sheet*

The Graph settings sheet allows you to enter a description of the graph you want to see, including the type of graph and the variables you want. The Graph settings sheet doesn't always contain the same settings—the particular settings depend on the type of graph you select. For example, there are

different settings for a bar chart than for a pie chart. Here is a list of the different kinds of Graph settings sheets.

- Bar/Line: For bar graphs, line graphs, stacked bar graphs, and percent bar graphs.
- Horizontal Bar: For horizontal bar graphs.
- Pie: For pie charts.
- XY: For XY scatter plots.
- High-Low-Close: Usually for displaying stock-market information (Javelin PLUS).

The type of graph you choose determines other settings on the settings sheet. For example, a pie chart does not have x- and y-axes, so these selections change when you select a pie chart (see below). When you create a new graph, the settings sheet starts out with bar/line settings.

All five Graph settings sheets contain the settings listed in Table 12.2. The rest of the settings differ depending on the graph type. They are explained when we discuss each type of graph.

To get started, make a bar graph of GNP. Select Bar as the Graph Type, and enter GNP as the first variable name in the Variables section of the settings sheet.

■ *Displaying a Graph*

Once you have specified this information, you can display the graph by pressing F10, the Graph key, which displays an updated version of the graph you most recently asked about. During a Javelin session, once you have given a / **V**iew **G**raph or / **D**efine **G**raph command, Javelin knows which graph you want to see and displays the last graph you referred to in one of these commands.

Press F10, and a bar chart of GNP appears (see Figure 12.2). Along the *x-axis* (the horizontal line along the bottom of the graph), Javelin identifies the time periods of the data, in this case years. Along the *y-axis* (the vertical line along the left side of the graph), the values for the variable are shown, to which the heights of the bars correspond. Notice that in order to avoid putting large numbers on the y-axis, Javelin has divided all numbers by 1000 and put the notation Thousands at the top of the axis. In a moment, you'll see how to change this (under "Scaling a Graph" below). There are no other legends, titles, or details yet.

Setting	Purpose
Name	You can change this to change the name of the graph. This name does not appear on the graph. It is used in Javelin commands, such as / **V**iew **G**raph.
Variables	Names of the variables to graph.
Legends	Text to display in the legend for each variable.
Labels	
Title	Text to display at the top of the graph.
Subtitle	Text to display under the title.
X-axis	Text to display as a label under the x-axis.
Y-axis	Text to display as a label above the top of the y-axis.
Graph Type	See Table 12.1 for the types of graphs.

Table 12.2: Settings common to all types of Graph settings sheets

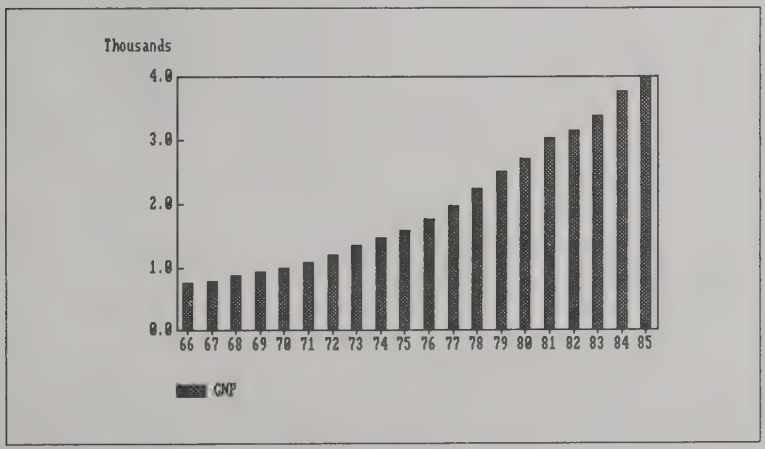


Figure 12.2: Bar graph of GNP

Now clear the graph from the screen by pressing Esc. You return to the / **D**efine **G**raph settings sheet, covering the other view(s) you had on the screen.

You can also display a graph by using the command / **V**iew **G**raph. Unlike the F10 key, this command prompts you for the name of a graph, so you can choose any graph you have already defined. As usual, the Spell key is available to help you choose among the graphs you have defined. The Graph key always displays the last graph you viewed or defined.

Editing a Graph

■ You can change the appearance of a graph by changing the settings on the settings sheet for it. The settings sheet for the graph GNP Current Dollars should still be on the screen (if it's not, give the command / **D**efine **G**raph and enter the name). Javelin will display the same Graph settings sheet you saw when you last defined the graph, and you can change any of the settings.

You can't change the relative heights of the bars or the positions of the lines on the line graphs—these represent the values of the variables in the graph. You have to change the values of the variable itself if you want the graph of the variable to change (use the Quick-graph or Chart view to enter and change data graphically).

■ Scaling a Graph

As you noticed above, Javelin automatically picks a scaling factor for the y-axis so that the data you are graphing will fit. In this case, it scaled the GNP figures into thousands. If you want to scale a graph explicitly, fill in the Max and Min values under the left and right axes (we'll talk more about the right axis in a moment). In this case, define Left Axis Max as 4000 and Left Axis Min as 0 by filling in those values. Display the graph again by pressing F10 (see Figure 12.3).

Now Javelin has removed its own label Thousands from the left axis and shows the numbers in the desired scale. But you should still label the axis as being in millions of dollars.

■ Legends and Labels

In order to identify the variables on the graph, the time periods covered, and the model from which the information comes, you can include text with your graph. There are two groups of settings that you can use, *legends* and *labels*. These settings are on all five Graph settings sheets.

The Legends and Labels settings are shown in Table 12.2. The Legends settings indicate the variables being plotted. If you leave the Legends settings blank, Javelin identifies the information on the graph by listing the variable names as they appear in the model. For each variable, you can enter a legend, which is shown on the graph instead of the variable name. For most graphs, the legends are displayed along the bottom of the graph.

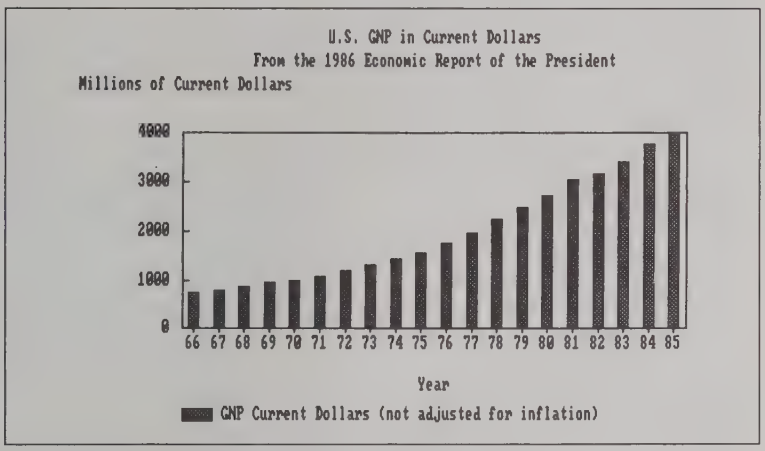

U.S. GNP in Current Dollars
From the 1986 Economic Report of the President
Millions of Current Dollars
4000
3000
2000
1000
0
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Year
GNP Current Dollars (not adjusted for inflation)

Figure 12.3: Bar graph with explicit scaling

Add titles and legends to the graph. For Title, enter

U.S. GNP in Current Dollars

and for Subtitle, enter

From the 1986 Economic Report of the President

For the x-axis and the y-axis, enter

Year

and

Millions of Current Dollars

Next, enter a legend to use on the graph rather than the default legend, which is the name of the variable (GNP). Under Legend, next to GNP, type

GNP Current Dollars (not adjusted for inflation)

Look at the graph again by pressing F10. Note where the titles and legends appear (see Figure 12.4). To leave the graph, press Escape.

■ *Numeric Formats and Dates*

Javelin uses its own judgment in showing numbers along the axes. The numbers displayed on the y-axis are not affected by the default number

format selected on the / **D**efine **S**ettings settings sheet. The only numbers whose format you can affect on a graph are those of the slices in a pie chart. Those numbers appear in whatever format the variable is defined to use (on the Variable settings sheet).

Usually, Javelin labels every point on the x-axis with its date. If there are too many dates to fit legibly, Javelin 1.1 labels just the beginning and ending dates; Javelin PLUS picks an interval so that the dates show legibly and labels every *nth* date (the data are still shown for every date).

The period of the graph is determined in one of three ways. First, if you give the graph beginning and ending dates, Javelin uses the time period you specify: if you begin in April 1987 and end in May 1987, your graph will be monthly; if you begin on 1 April 1987 and end on 1 May 1987, your graph will be daily. Second, if you do not specify dates but all the variables you are graphing have the same period, Javelin uses that period. Finally, if you don't specify dates and graph variables with different periods, Javelin uses the model default period. Rather than keep this somewhat complicated rule in mind, it is always good practice to tell Javelin explicitly the beginning and ending dates of your graph on the Graph settings sheet.

Let's look at each of the graph types and how to use them.

The Line Graph

■ The line graph shows up to seven different variables as lines connecting points that represent the values of each variable for each period. The line for each variable uses a different pattern or color, such as a solid line, a dotted line, or dashes, so that you can tell which is which.

Make a line graph comparing the unadjusted and adjusted GNP over time. The Graph settings sheet should still be on the screen; to change it to a line graph, change Graph Type to Line.

Enter another variable name to be graphed. Under Variables, below GNP, type

Adjusted GNP

Display the graph (see Figure 12.5). The title labels appear centered above the graph, the x-axis label is below the right end of the x-axis, and the y-axis label is above the top end of the y-axis. The legends or variable names

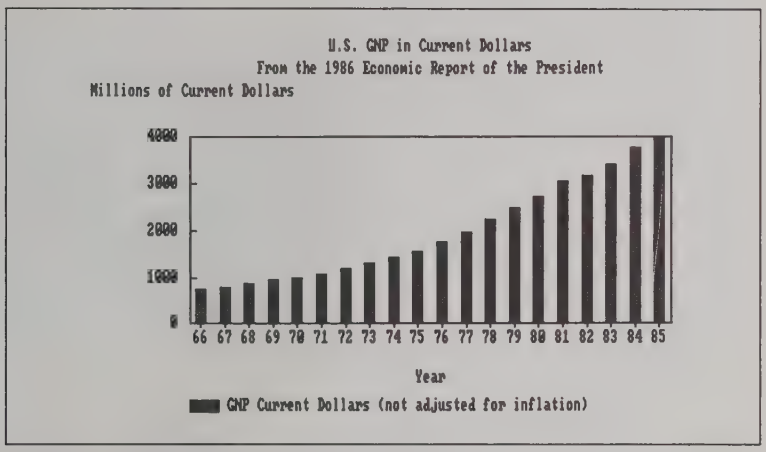

U.S. GNP in Current Dollars
From the 1986 Economic Report of the President
Millions of Current Dollars
4000
3000
2000
1000
0
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Year
GNP Current Dollars (not adjusted for inflation)

Figure 12.4: Graph with titles, legends, and labels

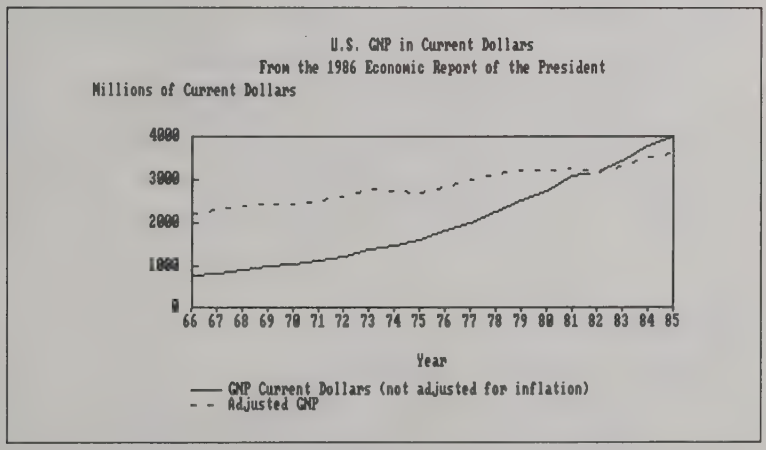

U.S. GNP in Current Dollars
From the 1986 Economic Report of the President
Millions of Current Dollars
4000
3000
2000
1000
0
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Year
GNP Current Dollars (not adjusted for inflation)
Adjusted GNP

Figure 12.5: Line graph of Current and Adjusted GNP

are displayed below the graph, with the line patterns used for each. Press Esc to return to the settings sheet.

■ *Right Y-Axis*

Sometimes you will want to graph variables that have different scales on the same graph. You can do this for line charts and bar charts in Javelin, by

using the *right axis*. For each variable to be graphed, you can choose whether it should be graphed against the scale on the left side of the graph or the right side of the graph. The maximum and minimum values for these two axes are completely independent, and you can set them on the settings sheet. The only thing you cannot do is provide a separate label for the right axis. For example, suppose you want to show growth in the GNP along with its size. The problem is that while the GNP size numbers are all between 1000 and 4000 million dollars, GNP growth is a percentage between −.1 and +.1. Obviously, you can't show them on the same scale, so use the right axis for the percentages.

For Right Axis Max, enter .1, and for Right Axis Min, enter −.1. Move down to an empty variable and enter

Pct Change GNP Adjusted

and select Right as the axis. The graph should look like Figure 12.6.

This graph uses two measures to make clear which axis each variable is plotted against: first, in the legend, include the axis that the variable uses, and in the y-axis label, include a note on what both axes are.

Right Axis works in bar graphs just as it does in line graphs. In order not to have too many different kinds of Graph settings sheets, Javelin also lets you specify a right axis in stacked bar graphs and percent bar graphs, but it ignores what you specify and graphs everything against the left axis. It's hard to imagine how a right axis could be used in a stacked bar graph, anyway.

The Bar Graph

■ This is the default graph type—the settings sheet for a new graph starts with Graph Type as Bar. For each period, each variable is shown as a vertical bar with the height representing the value for that period. The bars for each period are grouped together and shaded or colored differently for each variable so you can tell which is which.

Change Graph Type to Bar. The rest of the settings can remain the same. View the graph.

The first variable is the leftmost bar for each period, with the other bars following in order. The labels appear in the same places. The legends or variable names are displayed below the graph, with boxes showing the color or shading used for each variable. The graph is shown in Figure 12.7.

Notice that this graph is not really as clear as the equivalent line graph.

Line graphs are clearer if you want to emphasize the shape of the curve each variable makes (steep growth in Current Dollar GNP versus flat growth in Adjusted GNP), while bar graphs are clearer for comparing the values at each period. Line graphs are also clearer if your data run for many time periods. Let's demonstrate by creating a better bar chart. Clear the graph and the settings sheet by pressing Esc twice.

Start a new graph with the command / **D**efine **G**raph **F**ederal **S**pending. On this graph, you will compare federal spending on defense with non-defense spending. The comparison between the two figures for each year is

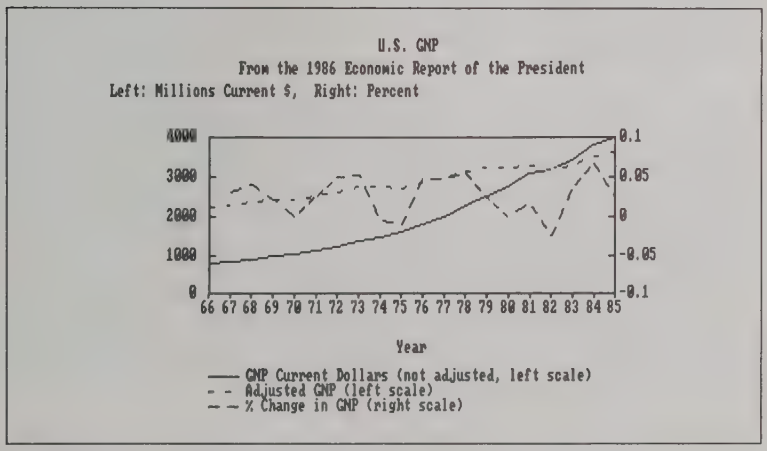


Figure 12.6: Line graph with right axis

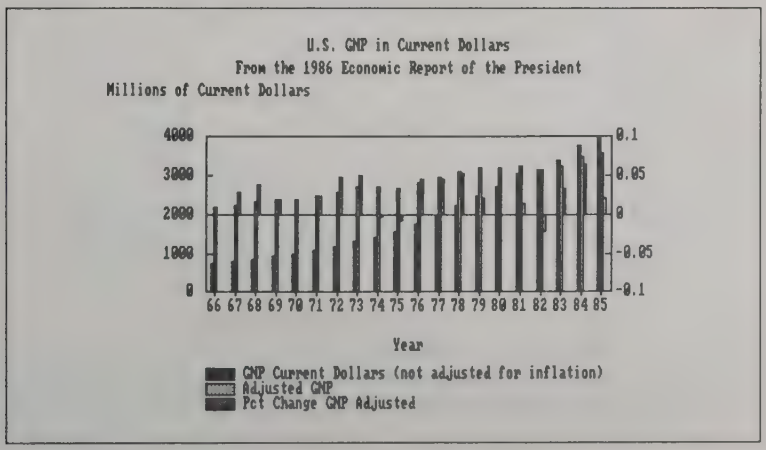


Figure 12.7: Bar graph of Current and Adjusted GNP

what is interesting. Enter the two variable names (use the Spell key to save typing):

Federal NonDefense Spending
Federal Defense Spending

The graph should look like Figure 12.8. Change it to a line graph and see if the graph is less effective visually. Change Graph Type to Line and redisplay it.

The Stacked Bar Graph

■ The stacked bar graph is like the bar graph, but the bars for each period are stacked up. This kind of graph is useful when the sum of your variables is a meaningful number, in this case, Total Federal Spending. The first variable (Federal NonDefense Spending) is on the bottom, with the second variable (Federal Defense Spending) just above it. The total height of the bar shows the total of the variables. There is one large bar for each period.

Change Graph Type to Stacked bar. The other settings can remain the same. This graph shows not only the values for Federal NonDefense Spending and Federal Defense Spending, but also the total of the two variables. The legends and labels are the same as in a normal bar chart. Let's add a title

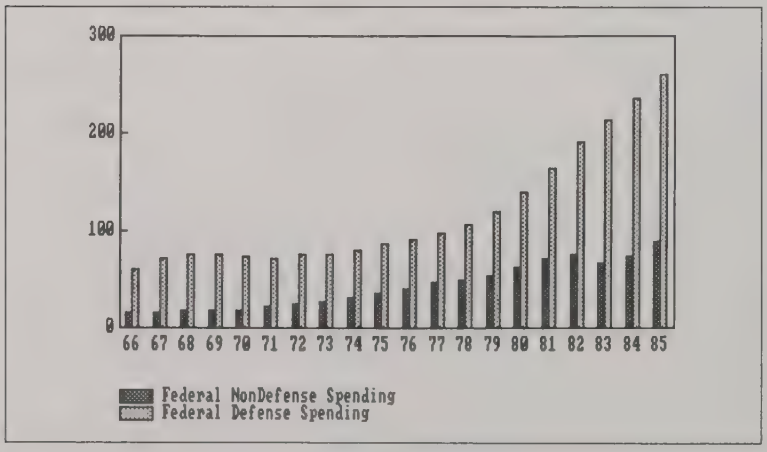


Figure 12.8: More appropriate bar graph

and some axis labels. For Title and Subtitle, type

U.S. Government Spending (Current Dollars)
From the 1986 Economic Report of the President

and for y-axis label, enter

Billions of Dollars

The graph is shown in Figure 12.9. The total height of the bars shows total federal spending for each year.

The Percent Bar Graph

■ You can show the makeup of a variable, that is, what other variables go into it in what proportions, with the percent bar graph. It works like the stacked bar graph, but the stacks are stretched or compressed so that they are all the same height. The graph shows what percent of the total stack each variable makes up; each stack adds up to 100 percent. There is one bar for each period.

A percent bar graph is probably the best way to compare defense and nondefense spending, showing what percentage of U.S. Government

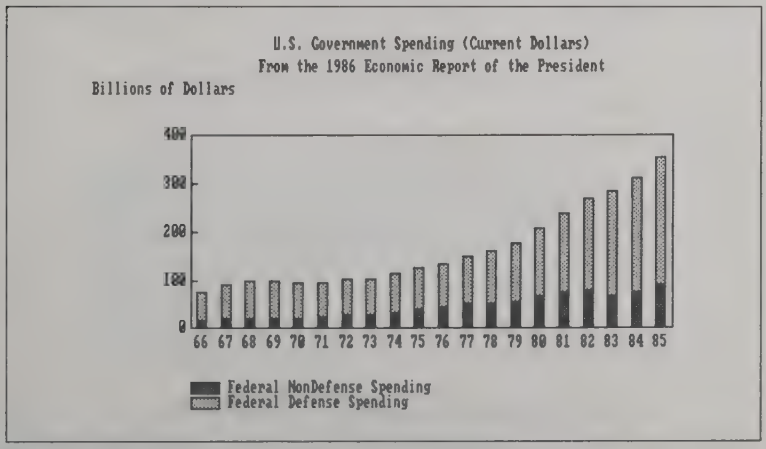

U.S. Government Spending (Current Dollars)
From the 1986 Economic Report of the President
Billions of Dollars
400
300
200
100
0
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Federal NonDefense Spending
Federal Defense Spending

Figure 12.9: Stacked bar graph of Defense and NonDefense Spending

expenditures go to defense versus nondefense programs. Change Graph Type to % bar and display the new graph.

The graph is shown in Figure 12.10. You could get the same result by creating two new variables, Defense Percent and NonDefense Percent, and graphing them as a stacked bar, and if you have some other use for these variables, the solution is just as good. But if you only need them to build the graph, let Javelin save you the trouble and use a percent bar graph.

Bar graphs also work if values are negative, and they do exactly what you would expect: the bars go below the line. The only two cases in which the result of a bar graph with negative data isn't obvious are in stacked bars and percent bars. Let's look at two such graphs, using data that is not in the model but that is related to the data you are using here: the U.S. trade balance on current account. It has two parts: trade in goods and trade in services. Although the U.S. service balance is typically much stronger than the goods balance, both have been positive and negative in the period 1966 to 1985.

Figure 12.11 shows a stacked bar chart where one of the values (usually Goods Balance) is negative. Javelin graphs the positive and negative bars both starting from the zero line. If both values are negative (as in 1967), Javelin stacks the negative values, just as it stacks the positive values. If one value is positive and another negative in the stacked bar graph, be careful what you assume about the meaning of the total height of the bar. Although in this graph you are looking at Goods Balance and Services Balance, the

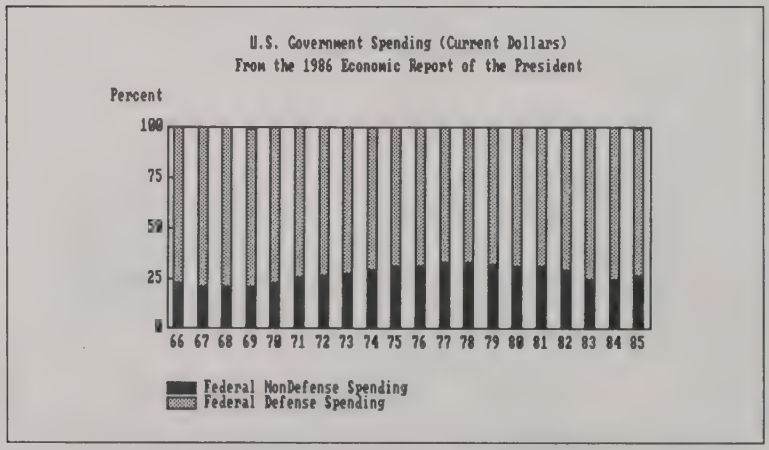

U.S. Government Spending (Current Dollars)
From the 1986 Economic Report of the President
Percent
100
75
50
25
0
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Federal NonDefense Spending
Federal Defense Spending

Figure 12.10: Percent bar graph of U.S. Government Spending

overall height of the bar is not the total U.S. trade balance. To show that, Javelin would have to *subtract* the height of the negative bars from the height of the positive bars, which it does not do. Instead, since it starts the positive bar from zero and the negative bar from zero, the total height is the volume of the trade imbalances in the U.S. economy—not a very meaningful number.

Figure 12.12 shows a stacked bar chart with negative numbers where the heights of the bars *are* meaningful. In this case, the height of the bar represents the volume of business that trade represented in the U.S. economy for the year. For example, in 1982 trade amounted to about $700 billion of the $3166 billion U.S. economy, or slightly over 20 percent of the GNP.

The moral of the story is that with stacked bars, particularly when some of your data are negative, think carefully about how to show your data to make your point.

The Horizontal Bar Graph

■ You may want to take a snapshot in time and show the situation for just one period. The horizontal bar graph displays up to seven variables for a single period as horizontal bars. The length of each bar indicates the value of the variable at that period. The topmost bar is the first variable listed on the settings sheet, and so on.

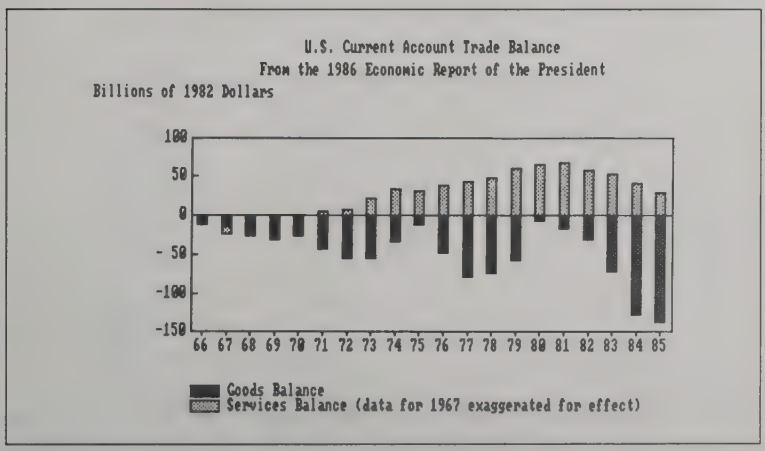

U.S. Current Account Trade Balance
From the 1986 Economic Report of the President
Billions of 1982 Dollars
100
50
0
- 50
-100
-150
66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85
Goods Balance
Services Balance (data for 1967 exaggerated for effect)

Figure 12.11: Stacked bars with negative values

Let's make a horizontal bar graph that shows various data about the U.S. in 1985 and call it 1985. Give the command / **D**efine **G**raph 1985. Set Graph Type to Horiz. bar. Notice that when you select this type of graph from the Graph settings sheet, several settings change on the sheet, as shown in Figure 12.13.

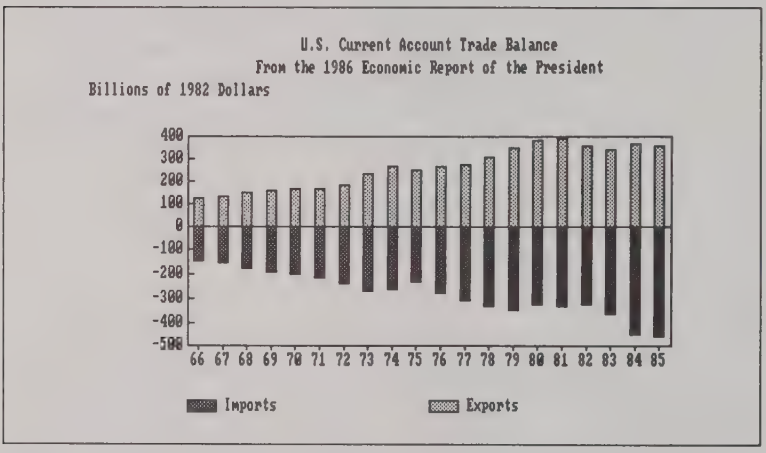


Figure 12.12: Stacked bars with negative values and meaningful heights

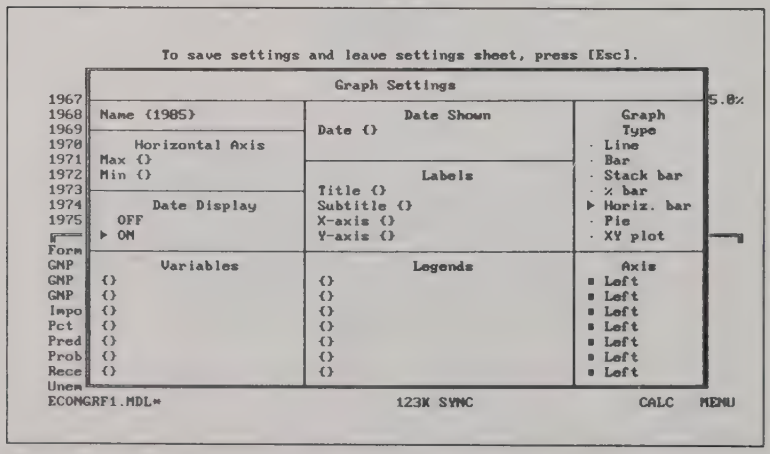


Figure 12.13: The Horizontal Bar Graph settings sheet

The additional settings are

- Horizontal Axis: Controls the scale of the graph; you can enter the maximum and minimum values to use on the x-axis.
- Date Display: Controls whether the name of the period is displayed at the bottom of the graph. You can turn this on or off; it is on unless you turn it off.
- Date Shown: Controls which period is displayed on the graph; you can enter the period. If you leave this blank, Javelin displays the first period for which any variable has data.

Enter the following variable names (these variables exist in your model, so use the Spell key to help):

Federal Spending
Federal NonDefense Spending
Federal Defense Spending
Imports

Set Date Shown to 1985. For Title, enter

U.S. GNP and Federal Spending

and for Subtitle, enter

From the 1986 Economic Report of the President

For the x-axis label, enter

Million Dollars

Display the graph. The bars are shaded or colored to distinguish them. The legends or variable names are listed down the side of the graph, identifying each bar. The labels appear in the same locations as in a normal bar graph, except since there is no y-axis, the y-axis label is ignored. The graph is shown in Figure 12.14.

As with other bar graphs, if some of your values are negative, they are placed below the x-axis (in this case, to the left). The values in a horizontal bar chart do not have to add up to anything meaningful, and in this case they do not. But it is a useful way to compare numbers of similar magnitudes (like imports and government spending).

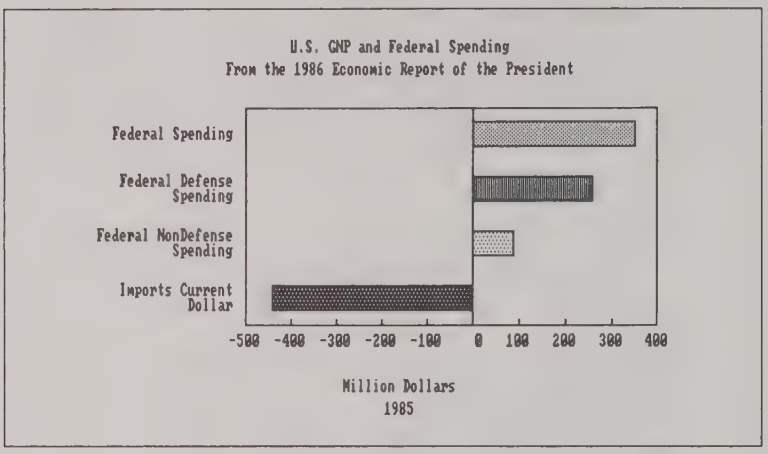

U.S. GNP and Federal Spending
From the 1986 Economic Report of the President
Federal Spending
Federal Defense Spending
Federal NonDefense Spending
Imports Current Dollar
-500 -400 -300 -200 -100 0 100 200 300 400
Million Dollars
1985

Figure 12.14: Horizontal bar graph of 1985 Government Spending

The Pie Chart

■ Like the horizontal bar graph, the pie chart shows up to seven variables for a single period. Like the percent bar graph, it shows the variables as a percentage of the total. Unlike either, the pie chart shows the values of each variable as slices of a pie (a circle) rather than as bars. The whole pie represents 100 percent. For this reason, the numbers you show on a pie chart should add up to something meaningful.

Modify your 1985 chart to make it a pie chart. Set Graph Type to Pie. When you select this type of graph from the Graph settings sheet, the additional settings for horizontal bar graphs disappear and are replaced by the additional settings for pie charts, as shown in Figure 12.15.

The additional settings are:

- Number Display: To control whether the percentages or actual values for each variable are displayed next to each pie wedge, turn this on or off. It is off for both % and Value unless you turn it on.
- Date Display: To control whether the name of the period is displayed at the bottom of the graph, turn this on or off. It is on unless you turn it off.

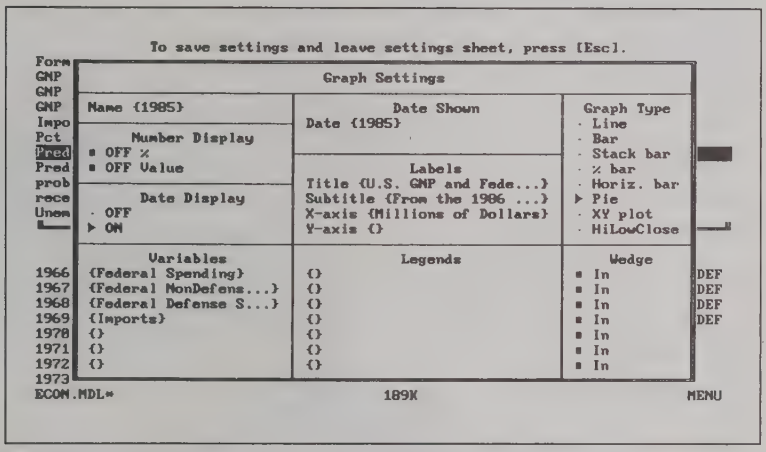


Figure 12.15: The Pie Chart Graph settings sheet

- Date Shown: To control which period is displayed on the graph, you can enter the period. If you leave this blank, Javelin displays the first period for which any variable has data.
- Wedge: To indicate whether the pie wedge for each variable is in or out of the pie. Pulling the wedge out of the pie emphasizes that value. Usually you will want to pull out only one variable. The setting for each variable is in unless you pull it out.

Display the graph; it should look like Figure 12.16. This is a good example of why the figures in a pie chart should add up to something meaningful; the sum of GNP, Federal Spending, Imports, Federal Defense Spending, and Federal NonDefense Spending is not a reasonable number. Negative numbers are ignored on a pie chart, since there is no reasonable way of showing them.

Fix this graph by deleting Imports, GNP, and Federal Spending, so that you graph only Federal Defense Spending and Federal NonDefense Spending, which add up to Total Federal Spending. Label the data a bit more explicitly, by setting Number Display–% and Number Display–Value to On. Finally, pull the wedge corresponding to NonDefense Spending out of the graph by selecting Out for the Wedge setting for that variable.

The graph is shown in Figure 12.17. The legends or variable names appear next to each pie wedge. The title labels appear at the top of the graph, but the x-axis and y-axis labels are ignored.

The XY Scatter Plot

■ All the graphs you have seen so far graph variables against time (line and bar graphs) or show variables graphically at one point in time (horizontal bars and pies). The XY Scatter Plot is different. Instead of showing a variable

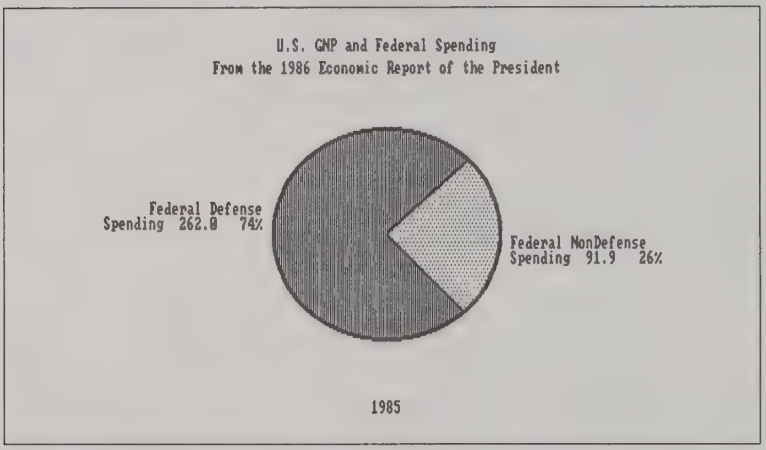


Figure 12.16: Poor example of a pie chart

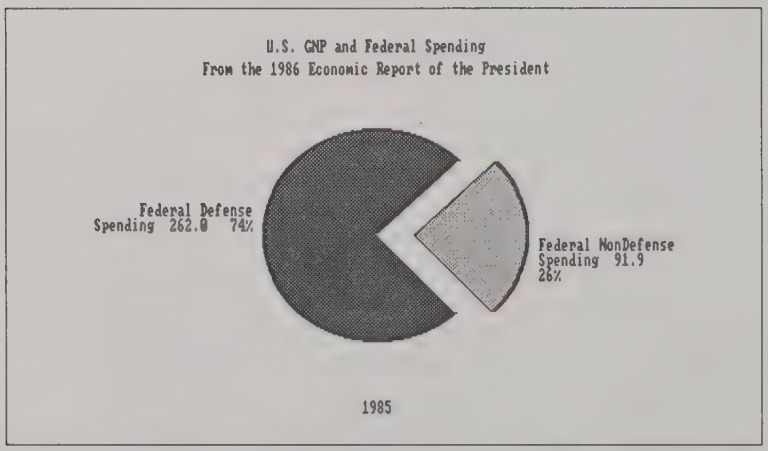


Figure 12.17: Better example of a pie chart

using time as the x value and the variable's value at time x for the y value, the XY plot uses *pairs* of values from two variables as x and y values, and puts a mark at that spot on the graph. In the XY plot, the time span controls which pairs of values (x from one variable, y from another) are used on the plot. XY plots are very useful to show whether there is a relationship between two variables.

This is actually a lot easier to look at than to explain; let's graph GNP Growth against Import Growth. If there is a relationship between GNP Growth and Import Growth, you would expect low GNP Growth rates to be associated with low Import Growth rates, and high GNP Growth rates to be associated with high Import Growth rates.

To define an XY graph, first clear the existing graph and Graph settings sheet by pressing Esc twice. Then give the command / **D**efine **G**raph and type

GNP vs Trade

Set Graph Type to XY and look at the settings sheet (see Figure 12.18). Not surprisingly, the information you need to specify an XY plot is quite different from the information you supply for the other graphs. Therefore the settings sheet is quite different.

The settings sheet shown here is from Javelin PLUS. There are two main differences from earlier versions: you may plot up to six different y variables

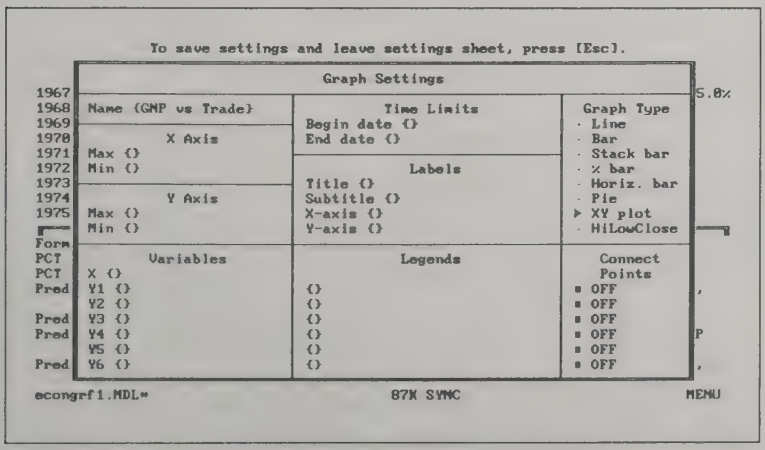


Figure 12.18: The XY Scatter Plot Graph settings sheet

against one x variable (explained below); and you have the option of connecting the points in time order (first to second to third) with a line.

In creating this graph, take the time to label the axes, because otherwise *there is no way to tell what is being graphed.* Unlike for other graphs, Javelin does not automatically show the x and y variable names on this graph. You must include them yourself in the axis labels in order to know what you are looking at. (Javelin PLUS shows the y variables in the legend area at the bottom of the graph but still does not label the x-axis.)

To compare the percent change of GNP to imports and exports, you need to create two new variables, Pct Change Imports and Pct Change Exports, to compare to Pct Change GNP. Exit from the settings sheet and type these formulas:

```
Pct Change Imports = PCT(Imports)
Pct Change Exports = PCT(Exports)
```

Recalculate the model so that these new variables will have values to graph, and redisplay the settings sheet using / **D**efine **G**raph.

Under Variables, for X, enter Pct Change GNP Adjusted, and for Y1 (Y in Javelin 1.1), enter PCT Change Imports. Under Labels, for the title, enter

GNP Growth vs. Trade

For the x-axis, enter

GNP Growth (Percent)

and for the y-axis, enter

Import Growth (Percent)

Figure 12.19 shows how this graph looks in Javelin PLUS. The Javelin 1.1 graph is almost identical, lacking only the variable legend at the bottom. Once again, since Javelin gives no way to format the numbers on the axes, you're stuck with what you see. You could create two new series, called something like PCT Change GNP Adjusted Scaled and PCT Change Imports Scaled, which would be the original series multiplied by 100, and then graph those new series. This would give you an x-axis running from −5 to +7.5 and a y-axis from −15 to +25. Those numbers look more like percents and might be clearer to your audience.

In Javelin PLUS, you can add additional series to your graph. Having seen how Import Growth relates to GNP Growth, perhaps you might want

Export Growth on the same graph. If you have Javelin PLUS, set the variable Y2 to PCT Change Exports and redisplay the graph.

Figure 12.20 shows the result. Both import growth and export growth appear to be related to GNP growth: they both grow faster when the GNP grows faster and more slowly when GNP grows more slowly. (See Chapter 15, "Statistics," for more information about determining correlations, as well as an example of Connect Points used to show a trend line on this graph.)

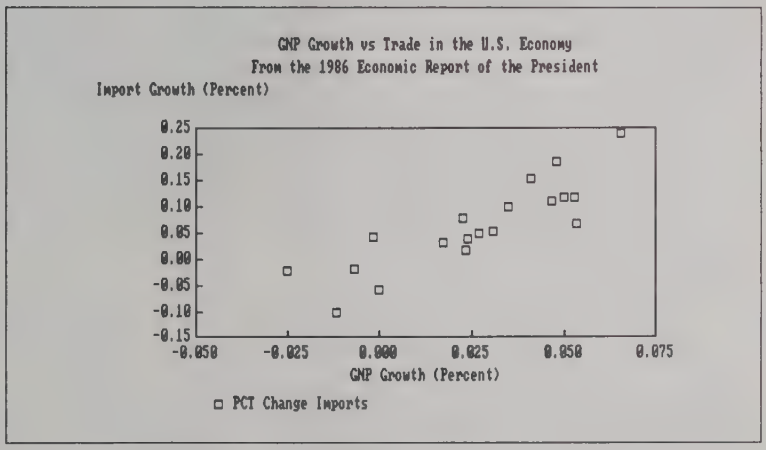


Figure 12.19: XY graph showing GNP Growth vs. Import Growth

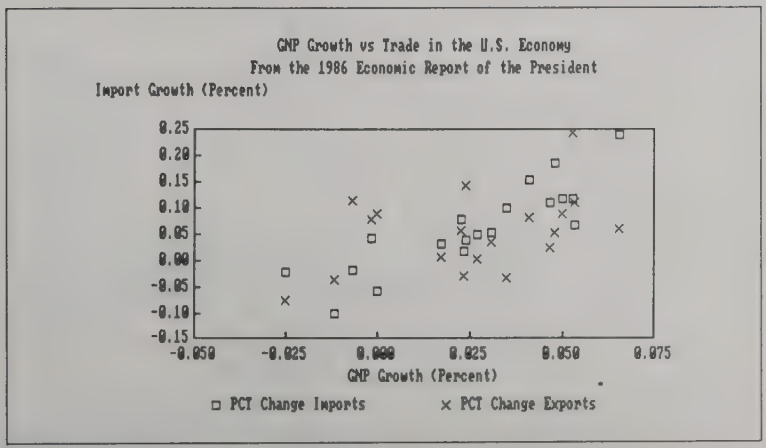


Figure 12.20: Javelin PLUS XY graph with two y variables

High-Low-Close Graphs (Javelin PLUS)

■ High-low-close graphs are a specialized tool of the financial industry; most people who need them know what they are. They present a large amount of information concisely because the information shown relates in a particular way.

The first three variables on the settings sheet are the highest price, the lowest price, and the closing price of a commodity or financial instrument during the period in question. These three variables are represented by a "floating" vertical line whose top is at the high price and bottom at the low price, and which has a tick mark at the closing price. These graphs often include a bar chart of the amount of the commodity sold (in stock market terms, the volume of shares traded). This is obviously on a different scale from the commodity price, typically hundreds or thousands of units. You may also include up to three additional series, which are drawn as lines on the graph.

Since a picture is worth a thousand words, Figure 12.21 shows an example of a high-low-close chart, using monthly stock prices as well as two other related series, volume and a three-month moving average of the closing prices. Note that if you do not label the chart with the name of the high-low-close series (usually in the title), the name of that data *will not appear*

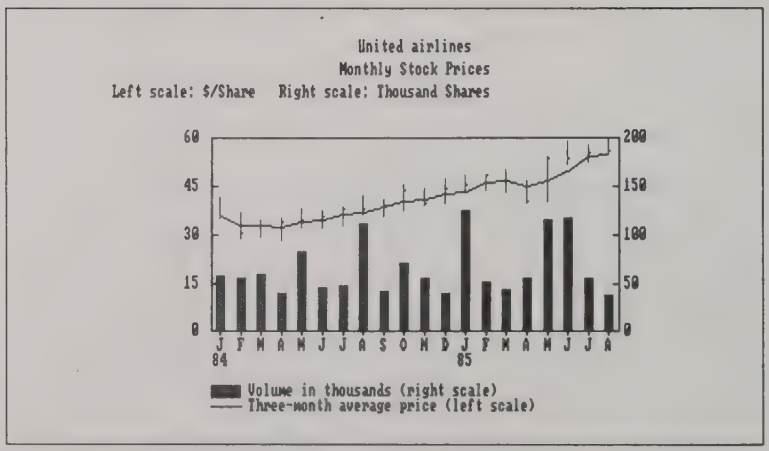


Figure 12.21: High-Low-Close graph of United Airlines Stock Price

anywhere on the chart. Since the high, low, and close series have three different names, Javelin cannot guess what name you want.

You will frequently use a right scale with this graph, especially if you include volumes. In addition, for a clearer graph, you will probably want to set both the left and right axis scales so that the floating high-low-close lines are near the top of the graph and the bars are well below them. Make sure to be clear about what your scales are and about which variable is against which scale. Figure 12.22 shows the settings sheet used to create the graph shown in Figure 12.21.

High-low-close graphs can also be used to combine one series of bars with up to three lines on a graph. This is the only way you can mix different representations of your data (lines and bars) on the same graph. Leave the high, low, and close variables blank and work with the rest of the settings sheet.

You have now seen all the types of graphs Javelin can produce. Let's see how to print them.

Printing a Graph

■ Printing graphs in Javelin is about as simple as can be: use the command / **P**rint **G**raph and indicate the name of the graph. Javelin prints the graph on

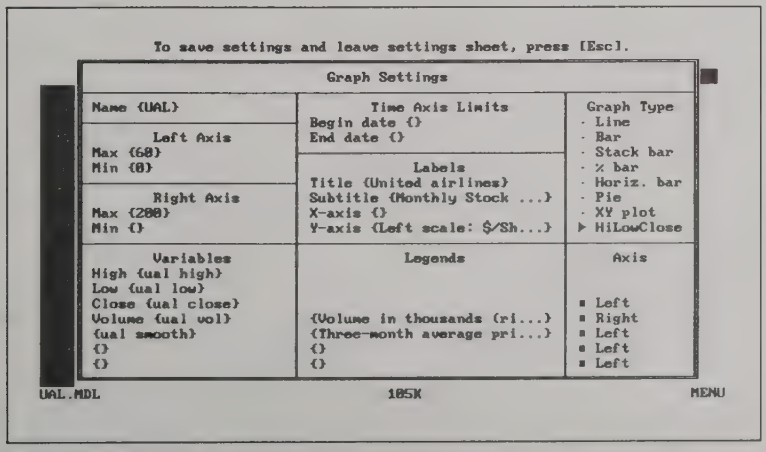


Figure 12.22: The High-Low-Close Graph settings sheet

the device you indicated when you installed Javelin. You must have a graphics printer (that is, one that can print dot graphics) or a plotter.

Make sure the printer or plotter is on, then give the command / **P**rint **G**raph. Type

GNP vs Trade

Figure 12.23 shows the graph as printed on an Apple LaserWriter printer.

When you print a graph, Javelin ignores the settings Page Limits, Header and Footer, Output Device, Setup String, and Miscellaneous on the Printer settings sheet. However, it redraws the graph to take advantage of all features of the printer or plotter you are using. Printed graphs may look better than they do on the screen, and the titles and legends may be aligned a little differently, as Javelin adjusts its layout for the printer's characteristics.

There are some things you might like to do when printing graphs in Javelin, such as specify where on the page the graph should go, how big it should be, or what colors should be used for what variables (on a plotter). But Javelin, which gives you a very simple printing facility for graphs, allows

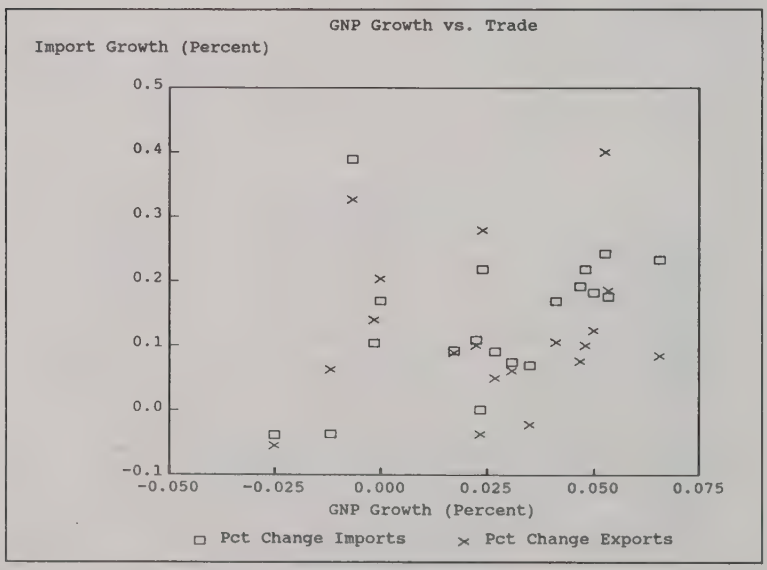


Figure 12.23: Graph as printed on Apple LaserWriter

no way to do this. To produce something fancier, look for a graphics program that allows you to capture the Javelin screens containing your graph and modify them for fancier graphics formatting. Such programs include MS-Windows Paint, GEM Paint, PC-Paint, and Freelance. Javelin PLUS can write industry-standard .PIC files, which you can read into many other graphics programs as well.

Tips on Creating Graphs

■ Don't try to jam too much information onto a graph. For example, a bar chart with seven variables is very difficult to read because it is hard to keep track of which bar represents which variable. Also, you can fit more bars on the screen with an Enhanced Graphics Adapter color monitor than with a regular Color Graphics Adapter, because the EGA has the detail necessary to show the graphs in color. (Javelin could show graphs in color in the CGA, but they would be so coarse as to be almost useless. CGA graphs are shown in black and white, but with more detail.) You can tell what color a small bar is more easily than you can identify the crosshatching pattern on a black-and-white graph. To be easily readable, an entire color bar chart shouldn't contain more than 20 bars, and a black-and-white chart should display a maximum of about 16 (a few more if you have a high-resolution printer like a laser printer).

When you are creating a graph, try switching to other graph types to see if there is a better way to represent the data. A virtue of the way the settings sheets work is that, although the settings change when you change graph types, *information is never lost.* If you switch from a line chart to an XY plot, your right-axis information, for example, will still be there when you return to a line chart. So feel free to experiment; it's very quick to do.

When setting up a pie chart, you may need to switch the order of the variables, so that when the chart is displayed the labels don't overlap. Javelin does its best to display the value and percentage clearly for each variable, but if you have several thin slices in a row, it may not be able to avoid having them overlap. In that case, change the order of the variables, and the order of the pie slices will change.

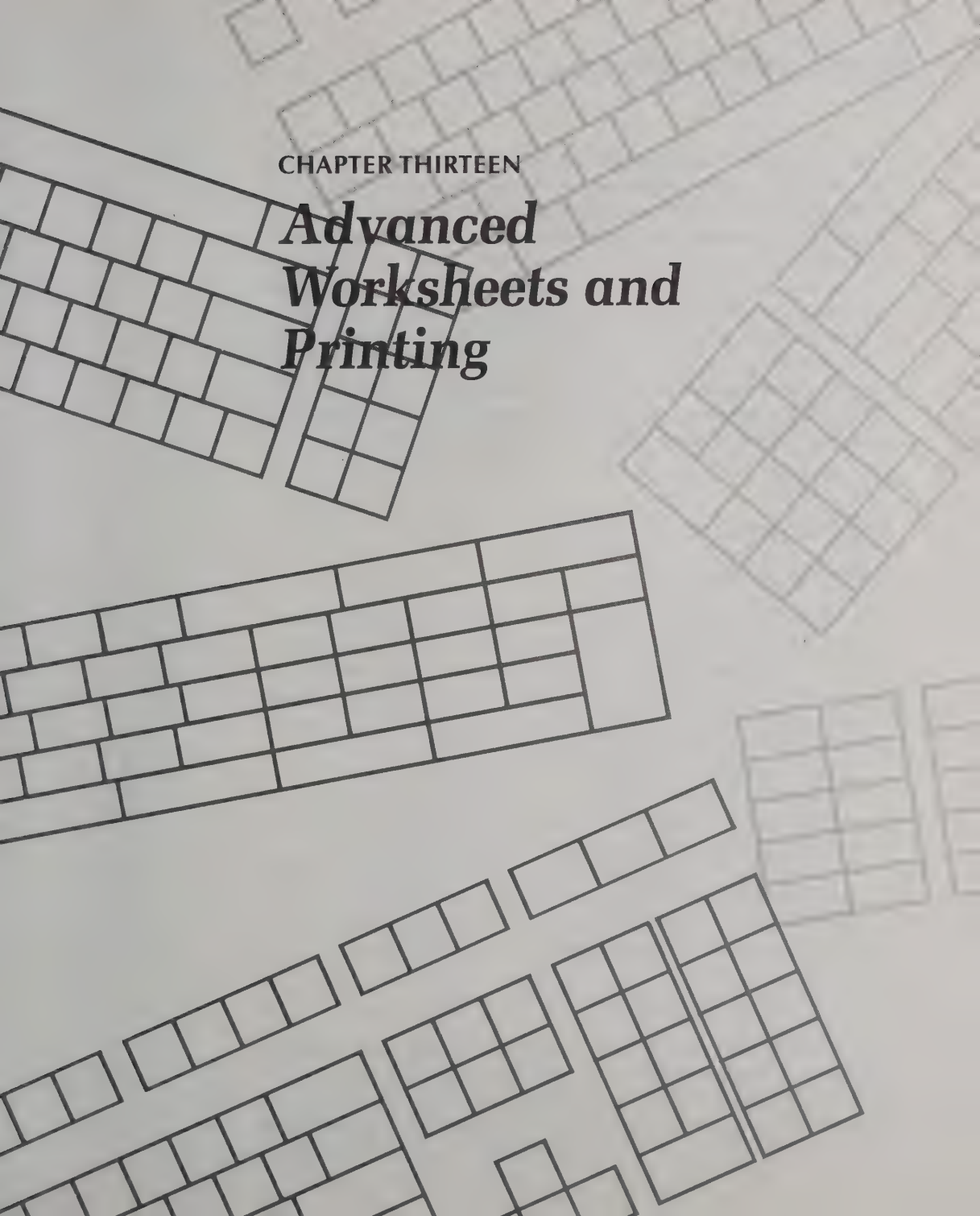
If you have a graph that you never want to see again, you can delete it from your model by using the command / **C**lear **G**raph.

Conclusion

■ Javelin has a fairly extensive set of graphs in its repertoire. Graphs represent another view of your underlying data, so that when the data change, the graphs change as well. As such, they are controlled by settings sheets, similar to the Variable and Worksheet settings sheets.

Different kinds of data lend themselves to different kinds of graphic presentation. Line graphs are better for showing the "shape" of a series of data over time, while bar graphs allow you to compare values at each moment in time. Stacked bars are useful when the total of your data is a useful number, while percent bars are good if you want to see how the composition of that bar changes over time. Horizontal bars and pies show different variables at a single point in time. While the variables shown in horizontal bars don't have to be related to one another, in pies the variables should all be breakdowns of some larger variable (such as defense and nondefense spending, unemployment by ethnic group, or trade by region). XY plots are useful for showing relationships between data, as well as for graphing whatever kinds of shapes you want to show. High-low-close charts are useful for plotting stock prices and others.

By putting such a rich set of graphing facilities at your disposal, Javelin encourages you to look at your data in new and clearer ways. Take advantage of it!



CHAPTER THIRTEEN

Advanced Worksheets and Printing

In Chapter 7, we discussed how to use Javelin's worksheets to create a model. In the last chapter, you saw how to dress up the data in your model into presentation-style graphs. Now you'll learn how to dress up worksheets into presentation-style reports, how variable names can make worksheet setup easier, and how to use modeling tricks in the Worksheet view. We'll start with the Worksheet settings sheet, talk about the / *Worksheet commands, and go into more depth on the use of borders. Then we'll show you how to use the Printer settings sheet to control the look of your reports.

This chapter uses two models. For most of it, you will work with a model that tracks a departmental budget. Later, to demonstrate worksheet sorting, you will use the 401K employee savings plan model from Chapter 10.

The Sample Model: Budget

■ Typically, a departmental budget has several major budget lines broken down into a number of detail lines. In the example, you will be concerned with only one major budget category (Salaries) and its breakdown into two minor lines (Exempt and NonExempt). Each line typically has a line number; call yours 10 for Salaries, 110 for Exempt, and 120 for NonExempt. In a complete budget, the other line items would work similarly.

For most budget categories, this breakdown would be sufficient. However, in a salary line you want to keep track of each salary in the department (even if you don't know which is whose), and for the nonexempt employees you want to track regular hours, overtime hours, and hourly rates.

Under 110 Exempt, therefore, you want a total of all exempt employees and a detail line for each. Under 120 NonExempt, you want not only the total but gross compensation for each employee, calculated as straight time plus time-and-a-half for overtime hours. So the formula looks like this:

Gross = Hourly * (Regular + 1.5 * Overtime)

Furthermore, for each line you want to track budgeted expenditures, actual expenditures, and the dollar variance between actual and budgeted. The model's goal is not only to analyze the information, but to make it convenient to enter new actual information as it is received and to generate reports.

Naming your variables properly is very important; the names should be hierarchical, so you can use worksheet headings, but they should also be easy to type using →|, the Spell key. For each budget category, you want budget, actual, and variance figures. In this example we have three exempt and three nonexempt employees. The budget variables are described in Table 13.1.

To create the Budget model, start Javelin with no model in memory. Create a worksheet called Input and enter the data and formulas from Figure 13.1. The finished Input worksheet is shown in Figure 13.2. All the variables in this model are monthly—to save typing, the data are shown for only January 1987. When you have entered the formulas and data, recalculate the model using F9 and save it as BUDGET.

Variable	Description
10 Salaries	Total of all salaries
110 Exempt Total	Total exempt salaries
110 Exempt 1	Salary for exempt employee 1
110 Exempt 2	Salary for exempt employee 2
110 Exempt 3	Salary for exempt employee 3
120 NonExempt Total	Total nonexempt salaries
120 NonExempt 1 Gross	Gross salary for nonexempt employee 1
120 NonExempt 1 Hourly	Hourly rate for nonexempt employee 1
120 NonExempt 1 Regular	Regular hours for nonexempt employee 1
120 NonExempt 1 Overtime	Overtime hours for nonexempt employee 1
120 NonExempt 2 Gross	Gross salary for nonexempt employee 2
120 NonExempt 2 Hourly	Hourly rate for nonexempt employee 2
120 NonExempt 2 Regular	Regular hours for nonexempt employee 2
120 NonExempt 2 Overtime	Overtime hours for nonexempt employee 2
120 NonExempt 3 Gross	Gross salary for nonexempt employee 3
120 NonExempt 3 Hourly	Hourly rate for nonexempt employee 3
120 NonExempt 3 Regular	Regular hours for nonexempt employee 3
120 NonExempt 3 Overtime	Overtime hours for nonexempt employee 3

Table 13.1: Variables for Departmental Budget

Variable	Formula or January Data
10 Total Salaries	110 Exempt Total + 120 NonExempt Total
110 Exempt	
Total	110 Exempt 1 + 110 Exempt 2 + 110 Exempt 3
1	3250
2	2825
3	4500
120 NonExempt Total	120 NonExempt Gross 1 + 120 NonExempt Gross 2 + 120 NonExempt Gross 3
120 NonExempt 1	
Gross	120 NonExempt 1 Hourly * (120 NonExempt 1 Regular + 1.5 * 120 NonExempt 1 Overtime)
Hourly	9.50
Regular	120.00
Overtime	5.00
120 NonExempt 2	
Gross	120 NonExempt 2 Hourly * (120 NonExempt 2 Regular + 1.5 * 120 NonExempt 2 Overtime)
Hourly	11.00
Regular	120.00
Overtime	5.00
120 NonExempt 3	
Gross	120 NonExempt 3 Hourly * (120 NonExempt 3 Regular + 1.5 * 120 NonExempt 3 Overtime)
Hourly	12.00
Regular	120.00
Overtime	10.00

Figure 13.1: Data and formulas for Input worksheet

```
10 Total Salaries
                                  Input
                          Jan 1987
10 Total Salaries        14,888.75

110 Exempt
  Total                  10,575.00
  1                       3,250.00
  2                       2,825.00
  3                       4,500.00

120 NonExempt Total       4,233.75

120 NonExempt 1
  Gross                   1,211.25
  Hourly                      9.50
  Regular                   120.00
  Overtime                    5.00

120 NonExempt 2
  Gross                   1,402.50
  Hourly                     11.00
  Regular                   120.00
BUDGET.MDL                    204K                          READY
```

Figure 13.2: Finished Input worksheet for budget variables

The Worksheet Settings Sheet

■ Each version of Javelin has added more settings to the Worksheet settings sheet to allow greater formatting flexibility (see Figure 13.3 for the settings sheet in Javelin PLUS). Table 13.2 explains what each setting does. All the settings control worksheet defaults; for example, the Column Width setting controls the default column width for all columns in the worksheet.

Display a Formulas view in the bottom window, and use F6, the Window key, to move to the top window. View a worksheet called Budget vs. Actual. To look at 1987, move to the top border above the first column in the worksheet and use the command / *Worksheet **B**order **D**ates to put the months of 1987 in the top border. For the begin date, type

Jan 87

and for the end date, type

Dec 87

You could see more columns if each column were narrower. The Worksheet settings sheet allows you to adjust the default column width for the whole worksheet. Use / **D**efine **W**orksheet to view the settings sheet for

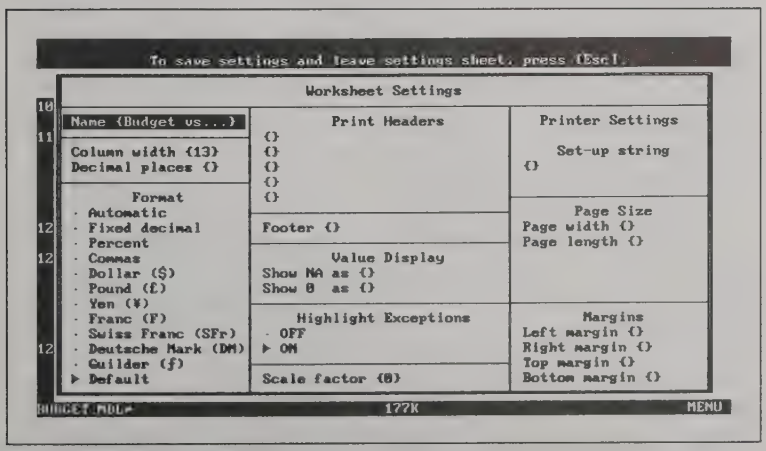


Figure 13.3: The Worksheet settings sheet

Setting	Purpose
Name	Name of worksheet, displayed at top of Worksheet view. You can change the worksheet name by editing this setting.
Column width	Default column width for all columns in the worksheet. You can override this for individual columns using / *Worksheet **W**idth (see below).
Decimal places	Default number of decimal places to show for numbers in worksheet. You can override this for individual rows, columns, or cells using / *Worksheet **F**ormat (see Chapter 7).
Format	Default format for displaying numbers. You can override this for individual rows, columns, or cells using / *Worksheet **F**ormat (see Chapter 7). The Default settings tell Javelin to use the model or variable defaults.
Header/Footer	(Javelin 1.1 and later.) Lines of text to print at top and bottom of each page when printing worksheet.
Display Value: Show NA as Show 0 as	(Javelin 1.1 and later.) How to display NA and zero values. Type in the labels you would like Javelin to display when a cell has the value NA or zero.
Highlight Exceptions	(Javelin 1.1 and later.) If On, highlight all exception cells in worksheet.
Scale factor	(Javelin 1.1 and later.) Multiply all numbers on worksheet by this scaling factor before displaying them.
Set-up String	(Javelin PLUS.) Special instructions to your printer before it prints. Default is set on Printer settings sheet.
Page width	(Javelin PLUS.) Number of characters your printer can print across the page from left edge to right edge. Default is set on Printer settings sheet.
Page length	(Javelin PLUS.) Number of lines your printer can print down the page. Default is set on Printer settings sheet.
Margins	(Javelin PLUS.) Left, right, top, and bottom margins to leave blank. Default is set on Printer settings sheet.

Table 13.2: Settings on the Worksheet settings sheet

Budget vs. Actual, set the default width to 9 (it starts at 13, the Javelin standard), and exit from the settings sheet. You can now see January through July on the worksheet. The left border has also narrowed.

■ *Column Widths*

Enter the first variable name for the worksheet in the left border of the first row of the worksheet border (not the corner); type

10 Total Salaries

The total appears for January. Although 9 characters is fine for data columns, the left border has to be wider. To change it, give the command / *Worksheet **W**idth.

Javelin prompts you with *Width:* and the current width of the column. Change the column width either by entering a new number or pressing ← and →. If you press ←, the column gets one character narrower. If you press →, it gets one character wider. Use these arrow keys until you like the looks of the column. Make the left border 18 characters wide, then press ↵.

Javelin requires that a column be one space wider than the width of a number to be displayed in it, so that the number won't run in with numbers in other columns. When displaying a number, Javelin uses the rightmost space only for the percent sign if you are using Percent format or for the close parenthesis if you are using a format in which negative numbers are shown in parentheses.

If you make the column too narrow for Javelin to fit the information in a cell, it responds in a way particular to the type of information in the column:

- Numbers: Displayed as asterisks if the column is not wide enough to display them in the format you have specified or in Automatic format. Change the format or widen the column.
- Labels: "Spill over" into the column(s) to the right of the column the label is in, as long as the cells to be spilled into would be blank otherwise.
- Dates: If a border column is too narrow to fit the normal day-month-year format Javelin uses, it will squeeze spaces out, leave off the year, and even leave off the month to make the date fit. In Javelin PLUS and later, you can have date information in body cells, and the squeezing works the same way (see Chapter 20, "Strings and Dates").

List the exempt employees first. Skip a row and in the left border, enter

110 Exempt

In the next border cell down, enter a space and type

Total

The total compensation for all exempt employees appears. In the next border cell down, enter a space and type

1

to indicate that you are interested in 110 Exempt 1. But when you press ⟵, Javelin thinks that you mean to type a date (see Chapter 9) and changes this to 2001. To indicate that this is part of a variable name and not a year, precede it with a number sign (#). Move up, leave a space, and type

#1

over 2001. Your 1 stays and the data for the variable appear. In the same way, enter left border cells for exempt employees 2 and 3.

Worksheet Range Calculations

■ If you display only some of the relevant data, the Worksheet view can be very misleading. On this worksheet, 110 Exempt Total is the sum of 110 Exempt 1, 2, and 3. If you add data for a fourth employee to the model and include rows for him/her on the worksheet, the formula for 110 Exempt Total won't be updated automatically to include the new variables. Or if you leave one of the employees off the worksheet, the worksheet will look wrong.

Delete the row for exempt employee 3 by moving to the left border for the row and pressing Del. Now the worksheet is confusing, because it looks as if 110 Exempt Total is supposed to be the sum of exempt employees 1 and 2.

In cases like this, where a variable really should be defined graphically, Javelin lets you do so. In this case, 110 Exempt Total should be defined as the sum of the detail lines directly below it.

You can use worksheet range calculations to define one variable as the sum, average, maximum, or minimum of a group of other variables

displayed on the worksheet. The other variables must be displayed together in a contiguous range.

The commands are:

- / *Worksheet **R**ange **S**um: Defines a variable as the sum of the values of other variables displayed in a range of the worksheet.
- / *Worksheet **R**ange **A**verage: Defines a variable as the average of the values of other variables displayed in a range of the worksheet.
- / *Worksheet **R**ange **H**igh/Max: Defines a variable as the maximum of the values of other variables displayed in a range of the worksheet.
- / *Worksheet **R**ange **L**ow/Min: Defines a variable as the minimum of the values of other variables displayed in a range of the worksheet.

In this example you will be working with a worksheet range sum, but the other three worksheet range calculations work the same way. This worksheet range calculation is defined as a series of rows, but range calculations also work when defined using columns.

■ *Defining a Worksheet Range Calculation*

To redefine 110 Exempt Total as a worksheet range sum, move the cursor to a body cell for that variable (move up to its January value) and use the command / *Worksheet **R**ange **S**um. Javelin asks you to point to the upper-left and lower-right corners of the range of variables to be included in the sum. In this case, the upper left is 110 Exempt 1, and the lower right is the blank line *below* 110 Exempt 2. Javelin won't let you include the variable you are defining in the range. That is, the row for 110 Exempt Total can't be included in the range to sum. If Javelin did let you do it, this would produce a circular reference and not the answer you want. (See Chapter 11 for information about circular references.) When you have highlighted the range, including the blank line, press ⏎.

Look in the formula window at the formula for 110 Exempt Total; this is a formula unlike any you have seen so far. Javelin has defined a *building block* whose name is the name of your variable (110 Exempt Total), the name of the worksheet you are using (Budget vs. Actual), and the legend Worksheet Sum. The inputs to this building block are 110 Exempt 1 and 110 Exempt 2. The output is 110 Exempt Total. When looking at a Formulas view, this name tells you everything you need to know to find out what the

formula does. (See Chapter 17, "Building Blocks," for a complete description of building blocks.)

Recalculate the model. The figure for 110 Exempt Total now includes just exempt employees 1 and 2 (see Figure 13.4).

To mark the end of the range sum, move over to the left border, below 110 Exempt 2 and enter a label like "end of sum" or "---". Any label will do, and it serves to indicate that 110 Exempt Total is the sum of items down to (and including) the label.

Why bother to define 110 Exempt Total with a worksheet range sum? Because now, if you insert a row between 110 Exempt 2 and the label, that value will automatically be included in the sum. With the cursor on the label, insert a row. In the left border, type a space and

```
#3
```

Not only does the value for 110 Exempt 3 appear, but you can see in the Formulas view that 110 Exempt Total has been updated as well. This is also why it is a good idea to end your worksheet range sum with a blank row, and to enter a label there. The label serves as a reminder of where your sum ends and where you should insert additional rows if you want to add values to the sum. The sum will always expand or contract as you insert or delete rows within its range. If you delete the top or bottom row, the row just below the old top or just above the old bottom becomes the new top or bottom row. This makes it very difficult (though not impossible) to add a new top or bottom row. The easiest way to add such a row is to redefine the worksheet sum.

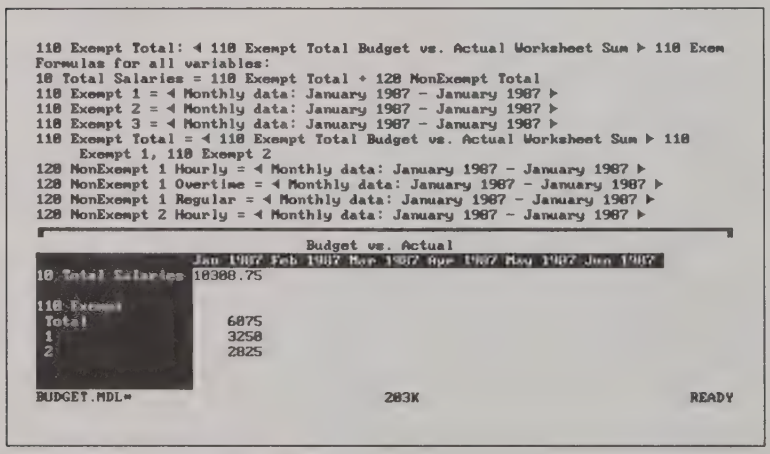

110 Exempt Total: ◄ 110 Exempt Total Budget vs. Actual Worksheet Sum ► 110 Exem
Formulas for all variables:
10 Total Salaries = 110 Exempt Total + 120 NonExempt Total
110 Exempt 1 = ◄ Monthly data: January 1987 - January 1987 ►
110 Exempt 2 = ◄ Monthly data: January 1987 - January 1987 ►
110 Exempt 3 = ◄ Monthly data: January 1987 - January 1987 ►
110 Exempt Total = ◄ 110 Exempt Total Budget vs. Actual Worksheet Sum ► 110
Exempt 1, 110 Exempt 2
120 NonExempt 1 Hourly = ◄ Monthly data: January 1987 - January 1987 ►
120 NonExempt 1 Overtime = ◄ Monthly data: January 1987 - January 1987 ►
120 NonExempt 1 Regular = ◄ Monthly data: January 1987 - January 1987 ►
120 NonExempt 2 Hourly = ◄ Monthly data: January 1987 - January 1987 ►
Budget vs. Actual
Jan 1987 Feb 1987 Mar 1987 Apr 1987 May 1987 Jun 1987
10 Total Salaries 10388.75
110 Exempt
Total 6875
1 3250
2 2825
BUDGET.MDL* 283K READY

Figure 13.4: Worksheet range sum

If a variable appears twice in a worksheet range calculation, it is included only once in the calculation. But showing a variable twice within the range is misleading and should be avoided.

■ Looking at a Worksheet Range

If you forget what range is covered by a worksheet range sum, Javelin 1.1 and later include the command / *Worksheet **R**ange **E**xamine to highlight the range. The Formulas view shows you the variable names, but it does not show what other rows (or columns) are included.

Move the cursor to a body cell for the variable defined by the worksheet range calculation, for example, the January value for 110 Exempt Total. Give the command / *Worksheet **R**ange **E**xamine. Javelin highlights the range of cells, including blank rows, so that you can check that the correct variables are included in the calculation.

If the range goes off the edge of the screen, you can press the period key to move the cursor from one corner of the range to the next (in 1-2-3 this is called *riding the range*). Hitting any other key makes the highlight go away.

If your worksheet is large, it may be difficult to find the variable defined by the worksheet range calculation, especially if you have displayed it with a label different from its name. Use the Goto key, F5, to find the variable on the worksheet.

■ Deleting a Worksheet Range Calculation

If you decide that you no longer want to define 110 Exempt Total as a worksheet range calculation, there are two ways to get rid of it: either delete all the rows that make up its range or clear the building block that represents it. The command / **C**lear **B**uilding block is used to delete a building block. It deletes the variable that was defined by the worksheet range calculation but does not affect any of the variables or rows in the worksheet range.

■ Advantages and Disadvantages

If your data are defined entirely by their relation on a worksheet, you should probably use worksheet range calculations. The fact that the formula readjusts as new rows or columns are added to the worksheet can help reduce errors and is very convenient. Worksheet range calculations work well for former spreadsheet users who are used to defining range sums, averages, maxima, and minima. On a spreadsheet, this makes sense, since

there is no good way to refer to values by variable names. However, since Javelin organizes all your data into variables with names, in most cases you will find worksheet range calculations more confusing to create and use than the SUM, AVG, MAX, and MIN functions (see Chapter 8).

There is one strange difference in Javelin 1.1 between the formula

110 Exempt Total = 110 Exempt 1 + 110 Exempt 2 + 110 Exempt 3

and the equivalent worksheet range sum. Whereas the formula gives you an answer only for time periods in which 110 Exempt 1, 2, and 3 *all* have data, the worksheet range sum gives an answer if *any* of the variables in its range has data. This most accurately simulates the actions of a spreadsheet program and generally gives the answer you want for graphically defined calculations. In Javelin PLUS, the formula gives the same answer as the range sum.

Copying Formulas and Data

■ Below the rows for exempt employees, enter the variable names for the nonexempt employees in the left border, as shown in Figure 13.5. Make sure that you indent the names as indicated, and use the command / *Worksheet **C**opy to enter labels that are duplicated. The data for the three nonexempt employees appear in the body of the worksheet. Tell Javelin to display only one window so you can see the whole worksheet, using / **W**indow **1**.

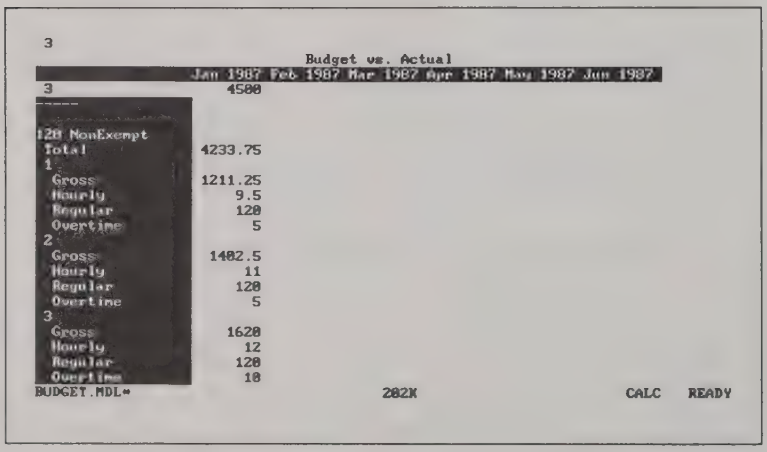


Figure 13.5: Labels for nonexempt employees

Most of the January salaries will remain constant all year, so the values can be copied from January to the rest of the months. Position your cursor on the first row of the worksheet, under Jan 87. Give the command / *Worksheet Copy. The range is from the current cursor position to the bottom of the worksheet. Anchor the cursor by pressing the period key. Move to the last row of the worksheet by pressing End and ↓, then press ⟵ to finish telling Javelin the range to copy from.

To copy these data into the succeeding eleven months, the range to copy to starts one cell to the right, so press → and anchor it by pressing the period key. Move to the last column in the worksheet by pressing End, →, and ⟵. Javelin copies the input data. The rows that are defined by formulas are unaffected. Recalculate the model, and data and results should appear for the rest of the year.

■ *Copying Formulas and Input Data*

Here's a full description of what / *Worksheet Copy does when you copy different parts of a worksheet:

- Dates in Borders: You can copy dates from one set of border cells to another. If you copy them into body cells, the dates are treated as labels. (In Javelin PLUS, they are treated as date values.)
- Variable Names in Borders: You can copy variable names or parts of them from one set of border cells to another. If you copy them into body cells, they are treated as labels.
- Labels: You can copy labels from anywhere to anywhere else. (In Javelin PLUS, they are treated as text values.)
- Exception Cells: You can copy exception cells from one set of body cells to another. You cannot copy exception cells into a border.
- Body Cells: When you copy body cells that display a variable, you copy the formula or input values for that variable. If you copy a body cell for a calculated variable listed under a heading to another variable listed under another heading, Javelin does a relative copy, replacing all instances of the words from the first heading with the words from the second heading (you did this in Chapter 7). If you copy a body cell containing a variable to a cell with no variable name in the border, Javelin copies the value, not the formula, and it becomes a numeric exception cell. If you copy a body cell into a border, nothing happens.

You can also use / *Worksheet **C**opy to copy from one worksheet to another. Display the worksheet you want to copy from in one window and the worksheet you want to copy to in the other. Start the copy. When asked to point to the place to copy to, use F6, the Window key, to jump from one worksheet to the other.

■ *Creating an Input Variable from a Calculated Variable*

Javelin has another copy command: / *Worksheet **D**ata copy. It is identical to / *Worksheet **C**opy, except if you are copying a formula / *Worksheet **C**opy copies (and rewrites) the formula, editing it to reflect the labels in the borders. / *Worksheet **D**ata copy *does not copy formulas.* Instead, it copies the numbers that are the results of the formulas. In other words, it is a convenient way to take the result of one formula and "type it in" as the input data to a different variable.

The command / **C**lear **F**ormula also converts a calculated variable to an input variable. It differs from / *Worksheet **D**ata copy because it converts a calculated variable into an input variable with the same name, rather than copying the input data into another variable.

For example, you might want to estimate the number of overtime hours the nonexempt employees will work, with the number increasing 5 percent each month over the course of the year. You can create a calculated variable called Overtime Estimate for the estimate you want to use and copy just the values into the overtime variable for each employee you expect to follow that pattern. You can then change the values for individual employees based on their schedules.

Move down several rows and enter the new formula into the left border cell. Type

```
Overtime Estimate = 10 @ Jan87; ROUND(PREVIOUS(Overtime
                    Estimate) * 1.05, 2)
```

This rounds off the number of overtime hours to two decimal places. Recalculate the model to see the answer.

Copy these values (but not the formula that created them) into the overtime budget values for nonexempt employee 3, using / *Worksheet **D**ata copy. For the range to copy from, highlight the values for General Overtime

Estimate from Jan 87 to Dec 87. For the range to copy to, highlight the January cell for 120 NonExempt 3 Overtime Budget. You can see that the values for the nonexempt employees' overtime budget are still input variables (see Figure 13.6).

■ *Renaming Groups of Variables*

Keeping track of employees by number is great for anonymity, but you might want to track the nonexempt employees by name instead. There are four variables for each nonexempt employee. Renaming each variable using / **D**efine **V**ariable would be tedious. Fortunately, with worksheets Javelin gives you a shortcut.

By using indented variable names in the left border (or superheadings on the top border; see below), you can change the names of all variables in the group at once, using the command / *Worksheet **N**ame change. In Chapter 7, you changed the name of one variable with this command. Now let's change the variable names of NonExempt 1 to Smith, NonExempt 2 to Jones, and NonExempt 3 to Feigenbaum.

Display two windows again, with / **W**indow **2**. In the upper window, show a Formulas view, and move down to look at formulas for 120 NonExempt 1, 120 NonExempt 2, and 120 NonExempt 3. In the lower window,

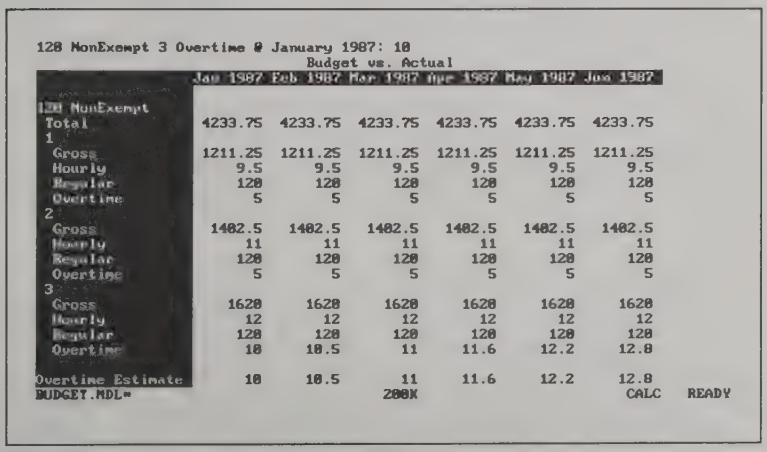


Figure 13.6: Worksheet after data copy

position the cursor on the left border variable name 1 (signifying 120 NonExempt 1) and give the command / *Worksheet **N**ame change.

The label 1 appears on the edit line. Whatever you put on the edit line now will replace the 1 in all variable names that this 1 in the border currently refers to. Delete the 1 by pressing Del, and type

Smith

The variable names in the Formulas view also change. Similarly, move down to the 2 and change that variable to Jones, and move to the 3 and change it to Feigenbaum.

Switch to the top window and display the Input worksheet. Move down and note that because you changed the name of NonExempt 1 to Smith, data for these variables are no longer displayed on this worksheet. The command / *Worksheet **N**ame change changes the names of the variables and changes the labels in the border of the worksheet you are using, but it does not update border labels in other worksheets in the model.

Fix the Input worksheet so that the data are displayed properly, changing 1 to Smith, 2 to Jones, and 3 to Feigenbaum in the left border. Switch back to the Budget vs. Actual worksheet.

To prepare for comparing budget to actual values, change the worksheet to show only the gross for each nonexempt employee. With the cursor in the left border, delete all rows below 120 NonExempt Total. In the border on the row below that, leave a space and type

Smith Gross

Below that, enter a space and type

Jones Gross

Finally, in the next left border cell down, enter a space and type

Feigenbaum Gross

Expand the window so you can see the whole worksheet, using / **W**indow **1**. Your worksheet should look like Figure 13.7.

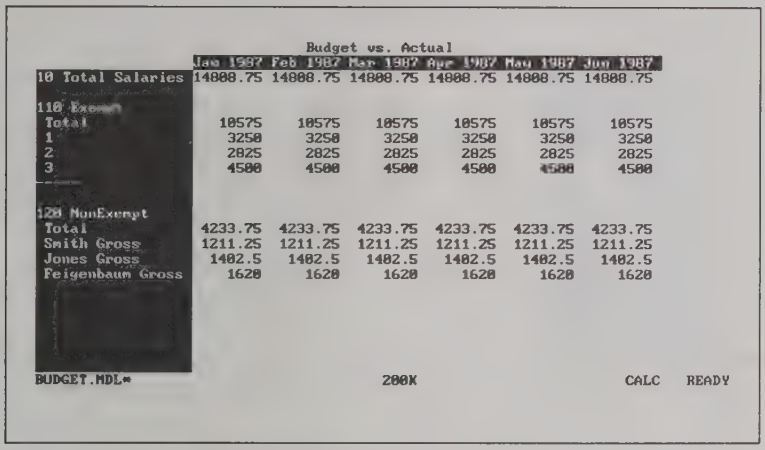


Figure 13.7: Budget vs. Actual worksheet with Budget totals

Using the Corner Border

■ The corner border, in the upper-left corner of the worksheet where the two borders intersect, controls all cells in the worksheet (unless you have additional borders; see below). Javelin decides what to display in each cell of the worksheet by combining the entries in the top border above the cell, the left border to the left of the cell, and the corner border. By typing an entry in the corner border, you create a "filter" that limits what will be included in every cell controlled by those borders.

Next, you want to enter actual information to compare with this budget information. In the corner border, type

Actual

Suddenly all the data disappear—none of the existing variables contains the word Actual in its name. Enter data for the actual values at January 1987 as in Figure 13.8; also enter the formulas for the actual total variables, as listed in Figure 13.9. You can also define Actual 110 Exempt Total and Actual 120 NonExempt Total as the equivalent worksheet range sums. Be sure to use F4,

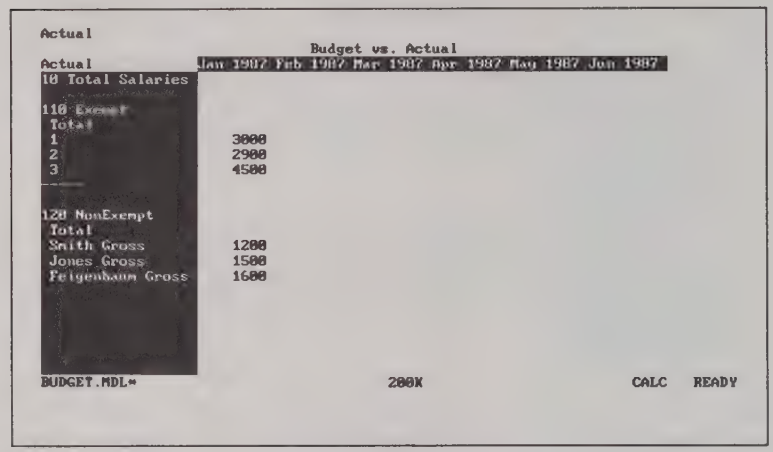


Figure 13.8: Actual figures to be compared to Budget

```
Actual 10 Total Salaries = Actual 110 Exempt Total + Actual 120 NonExempt Total

Actual 110 Exempt Total = Actual 110 Exempt 1 + Actual 110 Exempt 2 + Actual 110 Exempt 3

                        (or use the equivalent worksheet sum)

Actual 120 NonExempt Total = Actual 120 NonExempt Smith Gross +
                             Actual 120 NonExempt Jones Gross +
                             Actual 120 NonExempt Feigenbaum Gross

                             (or use the equivalent worksheet sum)
```

Figure 13.9: Formulas for Actual variables

the Point key, when entering these formulas, to save an enormous amount of typing. Note that the Actual variables don't exactly parallel the Budget variables—the Actual 120 NonExempt Gross variables are input variables, rather than calculated from hours and rates.

Multiple Borders

■ It would be interesting to compare the actual and budget figures for January. You can do this by adding a second vertical border to your worksheet. Up to now each worksheet has had one set of borders: one top border and

one left border intersecting in one corner border. However, Javelin lets you insert additional sets of borders below the first one, so that a different set of top borders is in effect for the lower part of the worksheet.

If a worksheet has several sets of borders, it separates the worksheet into several areas, each controlled by a set of borders. Each set of borders controls the body cells below and to the right of it, until the next set of borders. That is, a body cell is controlled by the first border cell above it, the left border cell to the left of it, and the first corner border cell above and to the left of it.

By using multiple borders, you can divide the worksheet vertically or horizontally. If the corner of the new set of borders is on the top border, the worksheet is divided vertically. If the corner of the new set of borders is on the left border, the worksheet is divided horizontally. The commands to handle multiple borders are / *Worksheet **B**order **I**nsert and / *Worksheet **B**order **R**emove.

■ *Inserting Another Border*

To compare actual and budget figures, insert a new set of borders to divide the worksheet vertically. The left set of borders will show Actual variables and the right set will show Budget.

Position the cursor at the cell where you want the corner of the new set of borders—in this case, in the cell of the top border that says Feb 1987. Insert a set of borders using the command / *Worksheet **B**order **I**nsert. Javelin inserts a column between Jan 1987 and Feb 1987, highlighted so that you know it's a border.

By entering variable names in it, you can compare actual values (in the body cells controlled by the left set of borders) to budget values (in the cells controlled by the right set of borders). Copy the column of variable names from the farthest left border to this new left border, from 10 Salaries down to Feigenbaum Gross. Be careful not to copy the word Actual in the upper-left corner. The Budget data appear.

Make the new left border 18 characters wide, using the command / *Worksheet **W**idth. Because the new border was inserted between the Jan 87 and Feb 87 columns, the first column it controls is Feb 87. Insert a column for Jan 87 just to the right of the new left border, in the Budget section of the worksheet.

Also take this opportunity to rename all budget variables to include the word Budget so that you can distinguish them from actual figures. In the upper-left corner of the Budget portion of the worksheet, use / *Worksheet

Name change to change the name in the corner border from blank to Budget. Look at the body cells—all the variables have the word Budget added to them (as in Figure 13.10).

An interesting thing happened to the worksheet range sum when you did this: 110 Total Exempt Budget is now defined by the Actual detail variables. Recalculate the model, and look at the results—there are data for January but for no other month. If you add up the numbers, they add up to the budgeted total, not the actual total. When you position the cursor on 110 Exempt Total, the formula displayed on the top of the screen refers to the Actual figures. You can also use the command / *Worksheet **R**ange **E**xamine to see what range 110 Total Exempt refers to.

There is a simple solution: use / *Worksheet **C**opy to copy the 110 Exempt Total formula from Actual to Budget. Or redefine it by giving the command / *Worksheet **R**ange **S**um again.

The use of borders is limited only by your imagination. In this case, you might want to add the variance variables to this worksheet, necessitating a third set of borders. Move to the Feb 1987 heading in the top border and insert a third set of borders, using the command / *Worksheet **B**order **I**nsert. In the new corner border, enter

Variance

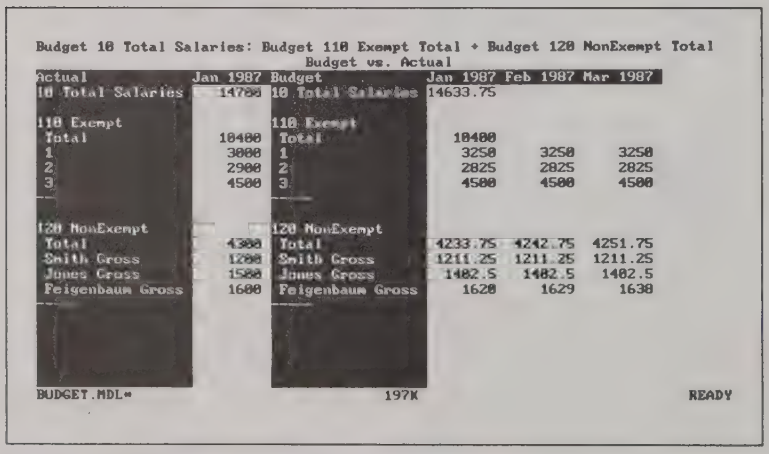


Figure 13.10: Budget variables after name change

Again, copy the variable names from the Actual or Budget left border to the Variance left border. Widen the Variance left border to 15 characters (if you make it 18, like the other left borders, the Variance Jan 1987 column won't fit on the screen). Insert a column for Jan 1987 to the right of the Variance left border.

To calculate the variances, subtract each Actual variable from the corresponding Budget variable. For Variance 10 Total Salaries, enter this formula (use the Point key to save typing):

Variance 10 Total Salaries = Budget 10 Total Salaries - Actual 10 Total Salaries

Copy this formula to the Variance rows for the 110 and 120 variables. Don't copy the formula to the rows with the --- labels or Javelin will create exception cells that you will have to delete. Your worksheet should look like Figure 13.11.

■ *Inserting Horizontal Borders*

You can also divide your worksheet horizontally. Move the cursor to the left Actual border, a row below the last variable, by pressing Home, End, ↓,

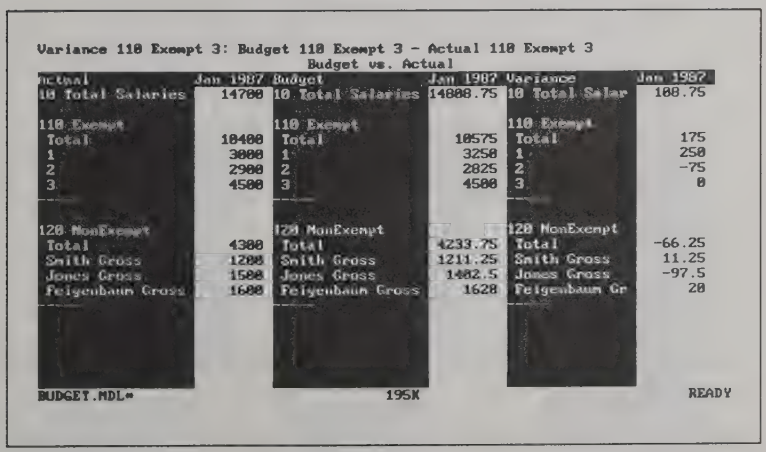

Variance 110 Exempt 3: Budget 110 Exempt 3 - Actual 110 Exempt 3
Budget vs. Actual
Actual Jan 1987 Budget Jan 1987 Variance Jan 1987
10 Total Salaries 14700 10 Total Salaries 14808.75 10 Total Salar 108.75
110 Exempt 110 Exempt 110 Exempt
Total 10400 Total 10575 Total 175
1 3000 1 3250 1 250
2 2900 2 2825 2 -75
3 4500 3 4500 3 0
120 NonExempt 120 NonExempt 120 NonExempt
Total 4300 Total 4233.75 Total -66.25
Smith Gross 1200 Smith Gross 1211.25 Smith Gross 11.25
Jones Gross 1500 Jones Gross 1402.5 Jones Gross -97.5
Feigenbaum Gross 1600 Feigenbaum Gross 1620 Feigenbaum Gr 20
BUDGET.MDL* 195K READY

Figure 13.11: Variance variables on Budget vs. Actual worksheet

and then ↓ several times. Give the command / *Worksheet **B**order **I**nsert. This time, because the cursor is in a left border, Javelin inserts a horizontal border.

You could use this section of the worksheet to do a summary, as shown in Figure 13.12. The Format setting on the Model settings sheet has been set to Commas to make it more readable.

Even with multiple borders, the width of each column is the same all the way down the worksheet, across additional top borders. If you need different column widths for the different parts of the worksheet, break it up into several worksheets instead.

■ Removing a Border

To delete a set of borders, use / *Worksheet **B**order **R**emove. Before you give the command, move the cursor to the corner border of the borders you want to remove. Remove the Variance section of the worksheet by moving the cursor to the corner border for the Variance figures, and giving the command / *Worksheet **B**order **R**emove. Also delete the extra Jan 1987 column.

■ Inserting Borders for Title Rows

You may want to have rows of information at the top of your worksheet, above the column headings. To do this, you can insert an additional set of

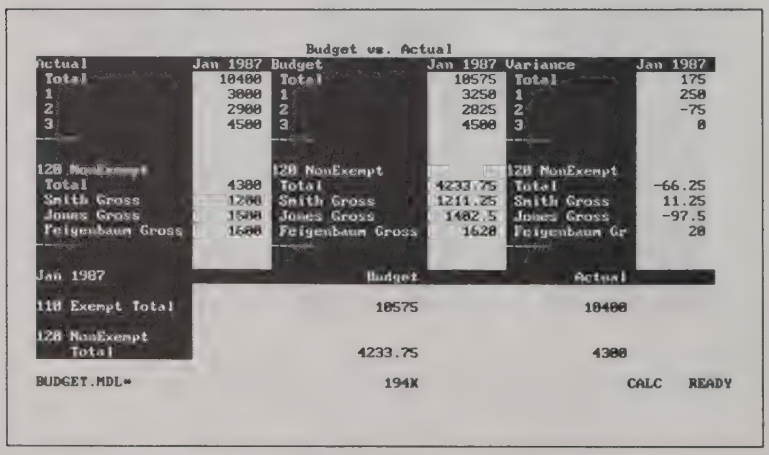

Budget vs. Actual
Actual Jan 1987 Budget Jan 1987 Variance Jan 1987
Total 10400 Total 10575 Total 175
1 3000 1 3250 1 250
2 2900 2 2825 2 -75
3 4500 3 4500 3 0
120 NonExempt 120 NonExempt 120 NonExempt
Total 4300 Total 4233.75 Total -66.25
Smith Gross 1200 Smith Gross 1211.25 Smith Gross 11.25
Jones Gross 1500 Jones Gross 1402.5 Jones Gross -97.5
Feigenbaum Gross 1600 Feigenbaum Gross 1620 Feigenbaum Gr 20
Jan 1987 Budget Actual
110 Exempt Total 10575 10400
120 NonExempt Total 4233.75 4300
BUDGET.MDL* 194K CALC READY

Figure 13.12: Summary section of Budget vs. Actual worksheet

borders above the existing top set of borders, to create blank rows at the top of the worksheet. To tell Javelin to put the new borders above the existing top set of borders, put the cursor in the top corner border when you give the command.

Press Home and give the command / *Worksheet **B**order **I**nsert. Insert a few rows in this area and type in some title information (see Figure 13.13). When printed, this title information is at the top of the first page only—to print information at the top of every page, use headers (described later in this chapter).

■ *Removing the Top Border*

You cannot use / *Worksheet **B**order **R**emove to delete the set of borders at the top left of the worksheet. If you move your cursor to the home cell and try to remove them, Javelin just beeps because removing this border would leave the rows below it without a top border.

In most cases, you can delete the top-left set of borders in your worksheet as follows. First, delete the contents of the border cells you want to get rid of. Then copy the border cells from the border that you want to be the top left border, using the command / *Worksheet **C**opy. Finally, delete the border that you just copied from. However, if you have an additional vertical border, as you do here, this won't work. You must remove the vertical border, move back up to the top, insert it again, and put back its variable names.

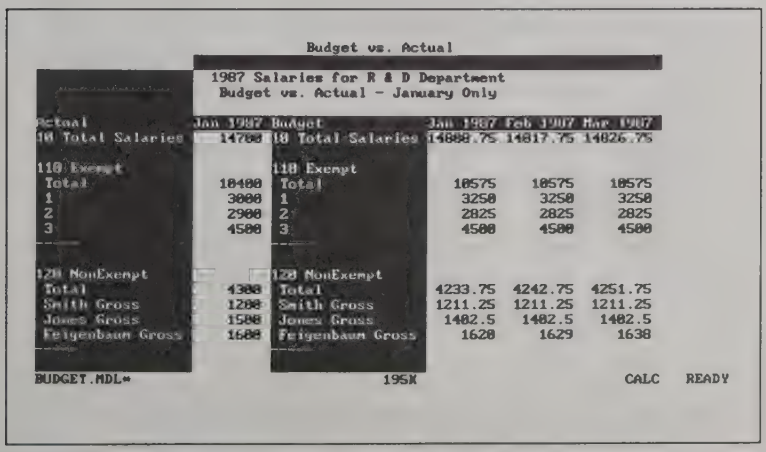


Figure 13.13: Additional worksheet heading information

Experiment with inserting and removing borders around your worksheet; you will find it a useful feature. But also remember that Javelin lets you have as many worksheets as you want in each model.

■ *Expanding and Shrinking the Top Border*

So far, you have done all your work with dates on the top border and variable names down the left border. Javelin treats the borders interchangeably, so there is no reason not to put variable names across the top and dates down the side.

Make a new worksheet, using / **V**iew **W**orksheet, and call it Comparison. Fill the left border with 1987 months using / *Worksheet **B**order **D**ates. In the corner border, enter

Budget

In the top border of the first column, enter

110 Exempt Total

and in the second column, enter

120 NonExempt Total

Your figures should appear.

If you have long variable names, you can make the top border several rows high to gain more space in which to type variable names. Two commands in the Worksheet star menu control the width of the top border, / *Worksheet **B**order **E**xpand and / *Worksheet **B**order **S**hrink.

With the cursor in the top border, give the command / *Worksheet **B**order **E**xpand. Now the top border is two rows high. Javelin combines the entries in the two border cells for each column to determine what's displayed in the column.

Move up to the cell containing 110 Exempt Total and edit it to remove the word Total. All the numbers disappear from the column. In the lower cell of the top border, type

Total

The figures reappear. Do the same for 120 NonExempt Total.

■ Superheadings

Suppose you want to see the detail information for Exempt Salaries 1, 2, and 3 alongside the total. Insert three columns between 110 Exempt Total and 120 NonExempt Total for this data. It would be tedious to have to type 110 Exempt in each column, and it wouldn't look good on the worksheet.

A *superheading* in the top border works the same way indenting works on the side border. First, type the part of the variable name into the topmost border cell of the first column. To make this heading extend across the next column to the right, move to its topmost border cell and press the hyphen key.

In the top border row of the second column, type a hyphen as the label. The heading 110 Exempt now extends to cover both columns. On the second row of the border, enter

```
#1
```

to indicate that you want data for 110 Exempt 1. Data appear.

You can extend this top level heading to cover employees 2 and 3 as well: use a hyphen as the top row heading and #2 (or #3) as the second row heading. Javelin centers 110 Exempt over the four columns (see Figure 13.14).

As when indenting labels in the left border, you can use multiple levels of superheadings in the top border.

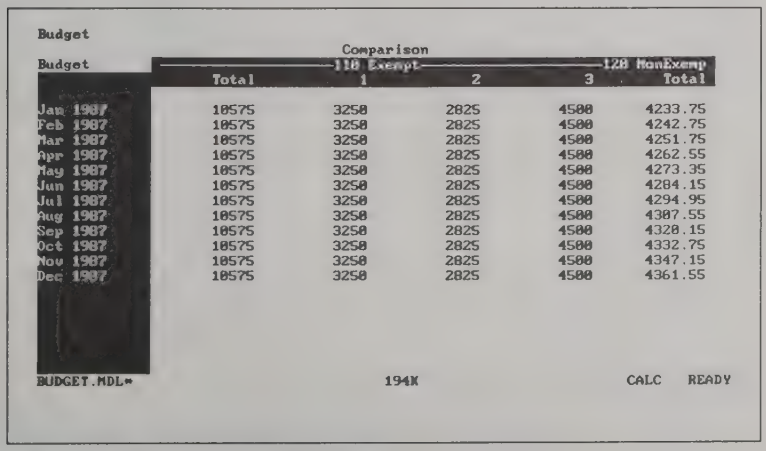


Figure 13.14: Superheadings in top border

■ Filling in Variable Names Automatically

Suppose you need a worksheet that contains only the total figures for the budget categories in your budget. By consulting a Formulas view, you could enter every variable with the word Total on the worksheet, but Javelin gives you another shortcut: the ability to automatically fill variables in the left border of a worksheet. / *Worksheet **B**order **V**ariables is similar to / *Worksheet **B**order **D**ates, but you can only fill variables into a left border, while / *Worksheet **B**order **D**ates works in either border.

View a new worksheet and call it Totals. In the heading of the first column, type

Jan 87

Move the cursor to the cell in the left border where you want the first variable name to appear. Give the command / *Worksheet **B**order **V**ariables. Javelin fills in all the variable names that exist in your model, in the order in which they were defined (see Figure 13.15).

■ Clearing the Worksheet

This isn't right—you want only variables whose names contain Total. Since you don't want all these variable names, you could delete each line of the left border, but take this opportunity to use the command / **C**lear **W**orksheet. This command not only clears the contents of the current worksheet but changes all the settings on its setting sheet back to Javelin's defaults. Give the command / **C**lear **W**orksheet. Javelin suggests the worksheet currently in view, Totals. Press ⟵ to clear it.

Give the command to view the Totals worksheet again. Once more, enter

Jan 87

in the top border of the first column.

■ Filling in Selected Variable Names

If you enter a heading in the left border, leave your cursor on the heading, and give the command / *Worksheet **B**order **V**ariables, Javelin lists only

the variable names that include the word(s) in the heading. The variable names are indented two spaces and the word(s) from the heading are omitted. Type

Total

in the left border of the first row of the worksheet. With your cursor on the word Total, give the command / *Worksheet **B**order **V**ariables. The variable names appear for both Budget and Actual, and Exempt and NonExempt.

This command also works with multiple levels of indented labels. Move to the left border cell just below Total and delete the ten variable names. Type two spaces and enter

Exempt

Without moving the cursor, give the command / *Worksheet **B**order **V**ariables, and just the Total Exempt variables appear. Below that, enter an indented heading NonExempt and do the same thing; the Total NonExempt variables appear. Make the column wider. Your screen should look like Figure 13.16. Save the model again.

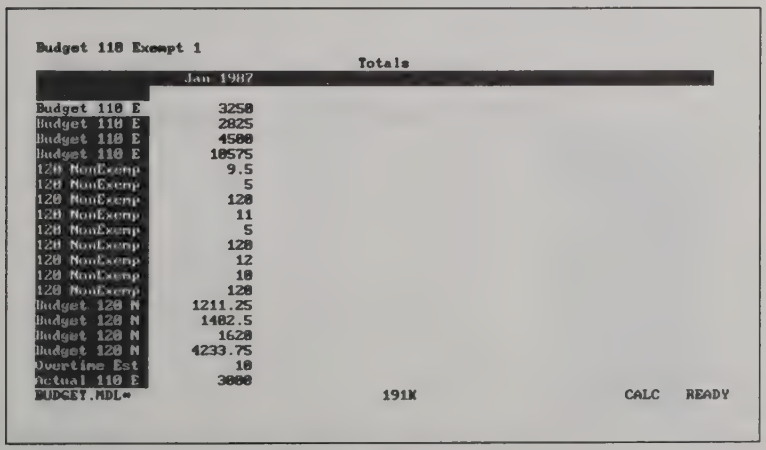


Figure 13.15: Variable names entered automatically

Reviewing the 401K Worksheet

■ For the rest of this chapter, you will use the employee 401K savings plan from Chapter 10. First, you will rank the employees by salary in an easier way. Then you'll use some additional border formatting features to create presentation worksheets for this model.

Load the 401K model. The input variables in this model are Salary, Rank, and Desired Contribution to the plan. Previously, you typed in the Rank for each employee yourself, ranking employees from 1 (highest paid) to 15 (lowest paid). This was easy with only 15 employees but would be time-consuming with 200. Using sorting, Javelin can assign the ranks for you. (In Chapter 14, "Macros," we will make this procedure even easier.)

View a new worksheet called Rank. Down the left border, use / *Worksheet **B**order **D**ates to fill employee numbers down the side, from 1 Emp to 15 Emp. In the top border of the first column, type

Salary

Salaries appear. In the next column, type

Rank

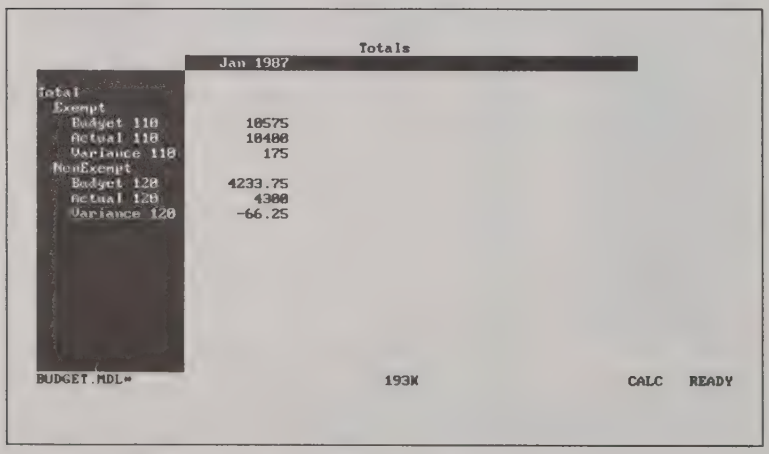


Figure 13.16: Totals worksheet with variables filled in by Javelin

The Rank values you entered before appear. To delete them, give the command / **C**lear **V**ariable to clear the Rank variable. Rank stays in the model because other formulas refer to it, but the input data are deleted and Rank becomes undefined (shown as #UNDEF on the worksheet). Now Javelin can fill in the rank of each employee for you.

You need a series of numbers from 1 to 15 to be used as the ranks, and Javelin can provide it. In the top border of the next column, type

```
Count = Since(1 Emp) + 1
```

Calculate the model. Javelin fills in Count as a number from 1 to the end of your Emp series (see Figure 13.17).

■ *Sorting Rows*

You want these rows to be in descending order by salary so that you can assign the numbers 1 to 15 to them. Javelin can sort the rows of your worksheet. There are many uses for sorting—for example, the command / *Worksheet **B**order **V**ariables lists the variable names in the order of their creation, which may not be the order in which you want to see them. You can sort the rows so that the names are in alphabetical order. Or you may want the variables listed in order by value, either lowest to highest or highest to lowest.

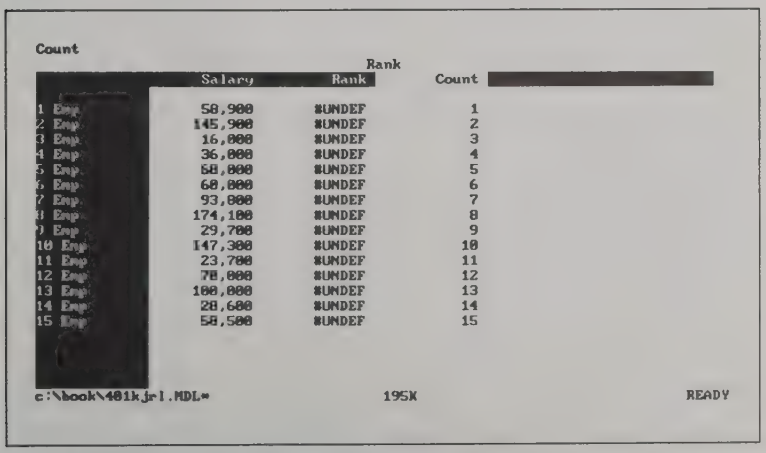


Figure 13.17: Count of employees in 401K model

When using the command / *Worksheet **S**ort, you must tell Javelin the following:

- Sort Direction: Sorts in ascending or descending order. For numbers, ascending is lowest to highest and descending is highest to lowest. For variable names or labels, ascending is alphabetical order and descending is reverse alphabetical order. For dates, ascending is chronological order and descending is reverse chronological order.
- Range to Sort: The range of rows you want to sort. The entire row, within the current borders, is sorted. The column you are in when you ask Javelin to sort is used as the *sort key* (that is, the value by which the rows are put in order).

Move to the Salary column for 1 Emp, since you want to sort on Salary. Give the command / *Worksheet **S**ort **D**escending. Select all 15 data rows, press the period key to anchor the cursor, move down to the end of the salaries, and press ⟵.

The whole width of the worksheet gets sorted—Dates, Salaries, Ranks, and Counts. Unfortunately, you want Counts to stay put, so the numbers 1 through 15 will be next to the employees in order by salary. Since sorting moves whole rows, the Count values are moved around with the data for the rows they were on.

Insert a second vertical border just to the left of Count. Move to the top border in the Count column and give the command / *Worksheet **B**order **I**nsert. All the values for Count disappear; those values exist only at dates, and so far your new border has no dates. Use / *Worksheet **B**order **D**ates to fill in dates at least from 1 Emp to 15 Emp in this new border, making sure to start on the same row that the dates start on in the farthest left border. Values for Count reappear.

Now the series from 1 to 15 parallels the series of salaries. You can turn these Count values into Rank values on the same row. Give the command / *Worksheet **D**ata copy to copy the rightmost Count column across the border into the Rank column. By using / *Worksheet **D**ata copy instead of / *Worksheet **C**opy, the actual values (1 through 15) are copied instead of their formula (Since(1 Emp) + 1): see Figure 13.18.

A final note on sorting: if you are using indented headings in the left border, Javelin sorts the rows *within* each heading. By using indented headings, you indicate that the groups of variables are to be displayed together. In Javelin PLUS, the whole groups are sorted so that the headings are in order, and then the detail rows are sorted within each heading.

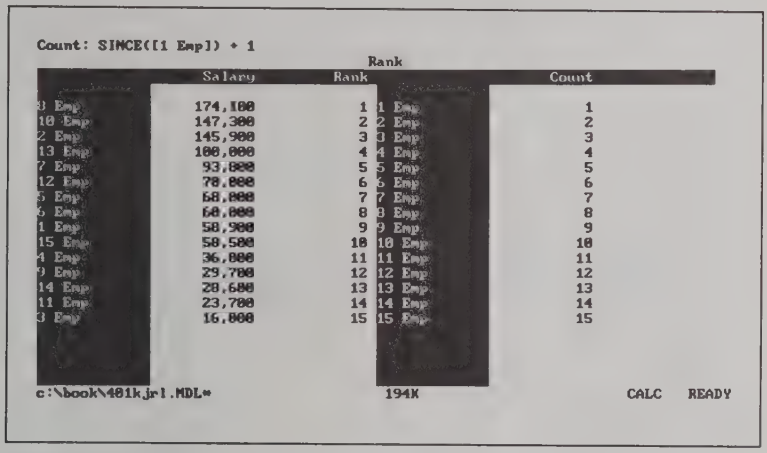


Figure 13.18: Rank values assigned by Javelin

Fine-Tuning the Appearance of the Worksheet

■ Several Javelin features help you fine-tune the appearance of your worksheet. In this section, you will apply them to worksheets in the 401K model.

■ *Scaling Numbers in a Worksheet*

In Javelin 1.1 and later, one setting on the Worksheet settings sheet is Scaling. If you enter a scaling factor, Javelin displays all numbers in the worksheet with the decimal point moved that many places to the right. For example, if your worksheet contains the number 123,456,789, here is how that number will look with different scaling factors:

- Scale factor −2: 12,345,678,900
- Scale factor −1: 1,234,567,890
- Scale factor 1: 12,345,678.9
- Scale factor 3: 123,456.789
- Scale factor 6: 123.456789

This allows you to show results in thousands or millions without having to change your formulas. You can also scale individual rows, columns, and cells on the worksheet by using the Format settings sheet, displayed with / *Worksheet Format.

View a new worksheet called Report and enter headings as shown in Figure 13.19. Expand the top border and adjust the column widths as needed.

The figures in the Salary column are all large numbers. To display them in thousands, move the cursor to the top border for the column and give the command / *Worksheet Format. Set the scale factor to 3. The last three digits disappear from the salaries.

When you use scale factors, it is important to include labels on your worksheet to indicate how the numbers have been scaled. You can't put a legend like Thousands in the top border pointing this out, because Javelin would think it was part of a variable name.

Putting a *label prefix* in front of text in the border tells Javelin that this text is a label, not a variable name.

■ *Label Prefixes*

There are four label prefixes that control where a label is displayed if it is narrower than the column it is in:

- ' (Apostrophe): Displays the label at the left side of the cell.
- " (Double quote): Displays the label at the right side of the cell.
- ^ (Caret; Shift-6): Displays the label centered in the cell.
- \ (Backslash): Displays the label repeatedly to fill the cell.

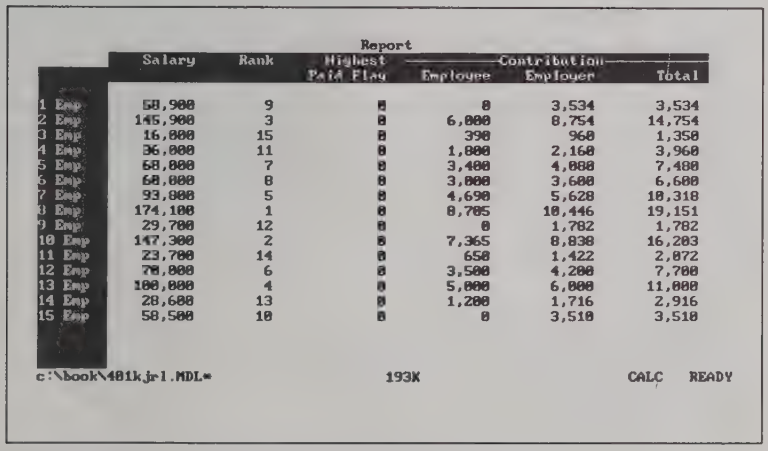
The backslash label prefix (also known as a *repeating label*) is especially useful for drawing lines. The most common repeating labels are \- and \=, which fill cells with single or double dashes.

When you type a label without a label prefix, Javelin automatically adds an apostrophe to the beginning of the label, positioning the label to the left. If you want to change the label prefix of an existing label, edit it using F2. In Javelin PLUS, you can also use the command / *Worksheet Justify to change the label prefixes for a range of cells.

Below Salary but still in the top border, enter

"(Thous.)

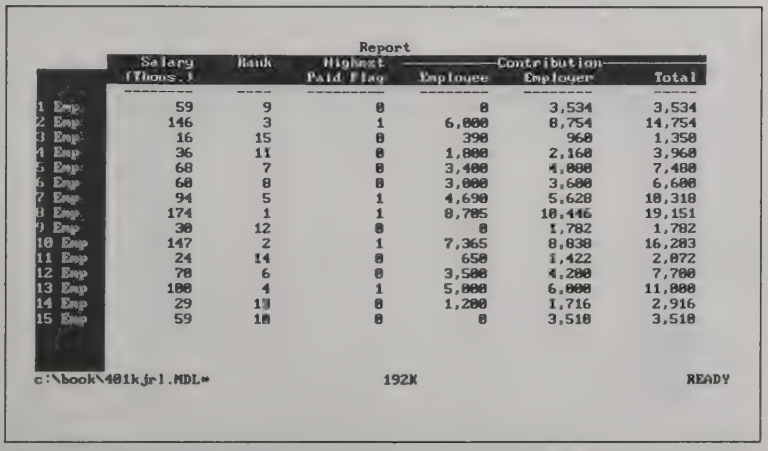
You do not need to close the double quote preceding the label. Put lines under the headings, in the first row of data cells, as shown in Figure 13.20.



	Report					
	Salary	Rank	Highest Paid Flag	Contribution Employee	Contribution Employer	Total
1 Emp	58,900	9	0	0	3,534	3,534
2 Emp	145,900	3	0	6,000	8,754	14,754
3 Emp	16,000	15	0	390	960	1,350
4 Emp	36,000	11	0	1,000	2,160	3,960
5 Emp	68,000	7	0	3,400	4,000	7,400
6 Emp	60,000	8	0	3,000	3,600	6,600
7 Emp	93,000	5	0	4,690	5,628	10,318
8 Emp	174,100	1	0	8,705	10,446	19,151
9 Emp	29,700	12	0	0	1,782	1,782
10 Emp	147,300	2	0	7,365	8,838	16,203
11 Emp	23,700	14	0	650	1,422	2,072
12 Emp	70,000	6	0	3,500	4,200	7,700
13 Emp	100,000	4	0	5,000	6,000	11,000
14 Emp	28,600	13	0	1,200	1,716	2,916
15 Emp	58,500	10	0	0	3,510	3,510

c:\book\401kjr1.MDL* 193K CALC READY

Figure 13.19: Report worksheet for 401K model



	Report					
	Salary (Thous.)	Rank	Highest Paid Flag	Contribution Employee	Contribution Employer	Total
1 Emp	59	9	0	0	3,534	3,534
2 Emp	146	3	1	6,000	8,754	14,754
3 Emp	16	15	0	390	960	1,350
4 Emp	36	11	0	1,000	2,160	3,960
5 Emp	68	7	0	3,400	4,000	7,400
6 Emp	60	8	0	3,000	3,600	6,600
7 Emp	94	5	1	4,690	5,628	10,318
8 Emp	174	1	1	8,705	10,446	19,151
9 Emp	30	12	0	0	1,782	1,782
10 Emp	147	2	1	7,365	8,838	16,203
11 Emp	24	14	0	650	1,422	2,072
12 Emp	70	6	0	3,500	4,200	7,700
13 Emp	100	4	1	5,000	6,000	11,000
14 Emp	29	13	0	1,200	1,716	2,916
15 Emp	59	10	0	0	3,510	3,510

c:\book\401kjr1.MDL* 192K READY

Figure 13.20: 401K worksheet with salaries scaled and labeled

■ Controlling the Appearance of Borders

When you have entered a variable name or a date in a border, sometimes you would prefer something else to be displayed in the cell. When you are typing a variable name, part of a variable name, or a date into a border cell, you can follow it with the label you want displayed in the border cell. You must separate the variable name or the date from the label with a label prefix.

For instance, instead of displaying Highest Paid Flag in the top border, which could be confusing, you can enter a label like *1 If In Top 3rd*. Don't display the variable name at all. If you press Del in the border cells, you delete the whole column. To delete the contents of the existing border cells without deleting the whole column, press Ctrl-D. Delete the contents of both border cells for the column by pressing Ctrl-D in each cell.

In the top border cell, type

Highest Paid Flag"1 If In

In the next border cell down, type

"Top 3rd

Your worksheet should look like Figure 13.21.

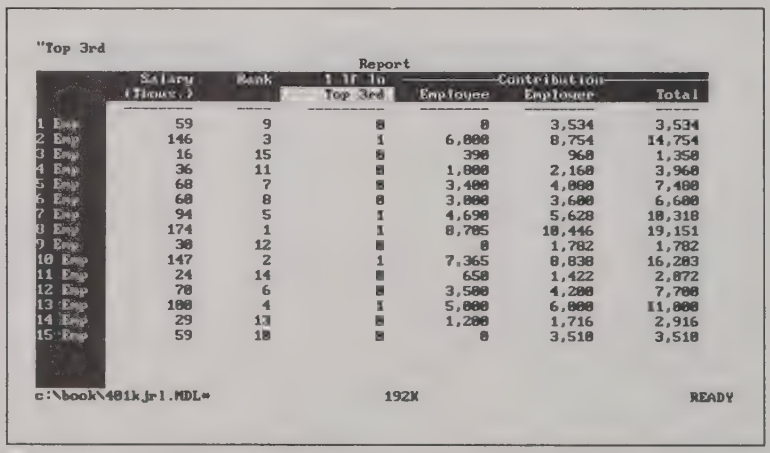


Figure 13.21: Worksheet with labels in top border

■ *Deleting a Range*

There are two / *Worksheet **R**ange commands that have nothing to do with worksheet range calculations. They are / *Worksheet **R**ange **D**elete and / *Worksheet **R**ange **P**rint.

/ *Worksheet **R**ange **D**elete allows you to delete, or clear out, a range of cells on the worksheet. The effect is the same as moving your cursor to each cell in the range and pressing Del. You can delete the contents of border cells or body cells. When deleting body cells, this only gets rid of labels and exception cells. What other body cells display is controlled by their top, left, and corner borders. / *Worksheet **R**ange **D**elete affects only what is shown on the worksheet: it doesn't delete any information from the central information base for your model.

View the Rank worksheet again. It contains a border and a column (Count) that you no longer need. To delete them, position the cursor in the top-left corner of the right border and give the command / *Worksheet **R**ange **D**elete.

Javelin asks you to highlight the area to be deleted. Anchor the cursor by pressing the period key. Then use the End and ↓ keys to go to the bottom of this border and the → key to highlight the Count column. When you press ⟵┘, the contents of the border and the column disappear. The border remains; remove it with / *Worksheet **B**order **R**emove.

■ *Printing a Range*

To print a whole worksheet, use the command / **P**rint **C**urrent view. But you may need to print out just a range of a worksheet—perhaps only the last three rows. To do this, use the command / *Worksheet **R**ange **P**rint and specify whether or not you want Javelin to print the border cells for the range you are printing.

View the Report worksheet again. To print it without including the salary column, you could delete the column and use / **P**rint **C**urrent view, but that would be unwise, since the names are not stored anywhere in the central information base; they are only labels on the worksheet.

Instead, give the command / *Worksheet **R**ange **P**rint. Highlight the range you want to print and press ⟵┘. You do not have to highlight the borders (particularly the left border in this case) because Javelin asks whether or not to include the borders on each page. Tell it to do so in order to tell which variable is which. Print just the Contribution columns.

The Printer Settings Sheet

■ To control the way Javelin prints your reports, change the settings on the Printer settings sheet, using the command / **D**efine **P**rinter. These printer settings are used whenever Javelin prints out a view, not just for printing worksheets. The only view for which Javelin does not use the printer settings is the Graph view. Table 13.3 lists the printer settings and what they do.

■ Using Setup Strings to Control the Printer

Most printers are designed to respond to special series of characters called *setup strings,* which control line spacing, boldface, underlining, and

Setting	Description
Output Device	General category of printer you are using. The default setting is *IBM character printer.*
IBM character printer	Choose this if your printer can print the IBM extended graphics character set. These include the line-drawing characters Javelin uses in the Diagram view.
ASCII character printer	Choose this if your printer cannot print the IBM extended graphics character set. Javelin will substitute normal characters wherever possible so that your printouts will look right based on the printer type you chose when you installed Javelin.
New file Append to file	Choose one of these if you want Javelin to store the text to be printed in a file rather than actually printing it (see Chapter 19, "Using Javelin with Other Programs").
Printer Port	How printer is connected to your computer. Parallel printers connect to LPT1, LPT2, or LPT3. Serial printers connect to COM1 or COM2. If you are using a serial printer, you may have to use the DOS MODE command before you start to print. The default setting is LPT1.
Setup String	Sends special instructions to your printer before it prints. Consult your printer manual for what setup strings it understands.
Limits	Size and shape of your paper and what margins you want it to use when printing.
Page width	Number of characters your printer can print across the page, from left edge to right edge. The default is 80.

Table 13.3: Settings on the Printer settings sheet

Setting	Description
Page length	Number of lines your printer can print down the page, from top edge to bottom edge. Most printers print six lines to an inch, so if you are using 11-inch paper, the page length is 66 (this is also the default setting).
Left margin	Number of spaces to leave blank at beginning of each line, to form a left margin. The default is 0.
Right margin	Number of spaces to leave blank at end of each line, to form a right margin. The default is 0.
Top margin	Number of lines to leave blank at top of each page, to form a top margin. The default is 3.
Bottom margin	Number of lines to leave blank at bottom of each page, to form a bottom margin. The default is 3.
Paper Saver	Affects printing of worksheets only, when printing a worksheet too wide to fit across a page. Javelin prints the worksheet in strips one page wide. If Paper Saver is Off, Javelin starts each strip on a new page. If Paper Saver is On, Javelin prints the strips without page breaks. This saves paper but makes it harder to tape the strip together to see a large worksheet. It may also ease the use of sideways printing programs. The default is Off.
Skip page after printing	Whether to skip to top of next page each print command. The default is On (so it does skip).
Pause between pages	If this is On, stops after printing each page and prompts you to put a new sheet of paper in printer. This is useful for feeding single sheets into printers with no mechanism. The default is Off.
Header/Footer	Text to be printed at top or bottom of each page.
Parameters	(Javelin PLUS) If you have a serial printer attached to COM1 or COM2, you can set the baud rate, parity, stop bits, and data bits for your printer. The possible settings for these parameters are the same as the ones for the DOS MODE command, as explained in the DOS reference manual.
Graph Port	(Javelin PLUS) If you print your graphs on a different printer or plotter than the one you use to print your reports, you can specify the port to which it is connected here. The possible settings are the same as the ones for the Printer Port.
Graph Parameters	(Javelin PLUS) If you print your graphs on a serial printer attached to COM1 and COM2, you can set the port parameters here. The possible settings are the same as for the Parameters described above. If you use the same serial printer for reports and graphs, the Parameters and Graph Parameters should be set the same.

Table 13.3: Settings on the Printer settings sheet (continued)

other qualities. Unfortunately, different printers have different capabilities; there is no standard for what feature is controlled by what setup string. You must consult your printer manual to find the setup strings your printer understands. We will use setup strings that work for Epson and compatible printers.

Setup strings can include normal alphabetic letters and digits as well as control characters. Control characters include the characters that tell the printer to begin a new line or begin a new page. Control characters can be represented by their character code, which is a number between 0 and 255. To make matters worse, the codes may be listed as decimal (base-10) or hexadecimal (base-16) numbers. To use a control code in your setup string, you must know the code for the character, expressed as a decimal number.

To type a setup string into the Printer settings sheet, you type in letters and digits normally, and you type control characters by using a backslash followed by the decimal code. Begin the code with one or more zeros if needed so that the code is three digits long.

For example, in the Epson printer manual, the setup string for condensed printing is code 15 (decimal), also called Ctrl-O. To specify this as your setup string, you would type

```
\015
```

and press ⟵ to get the setup setting. Because the printer can fit more characters across the page in condensed print, make the page width wider as well (described below).

To stop printing in condensed mode, you must do two things. First, remove the setup string from the Printer settings sheet, so that Javelin won't send the code to the printer again. Second, tell your printer to forget the code it received before. Most printers are designed to continue printing in the way you specified until receiving another setup string. To make the printer forget about condensed printing, send it the setup string for *cancel condensed print* (\018 for Epsons), give the command / **P**rint **A**lign, which resets the printer, or turn the printer off and on again.

Another code used often is Escape. You cannot type this into your setup string by pressing the Esc key, because Javelin interprets this to mean you want to leave the settings sheet. The code (decimal) for Escape is 27, so to include an Escape code in your setup string, type

```
\027
```

Epsons interpret the Escape code followed by the digit 0 to mean *print 8 lines rather than 6 lines to the inch.* You would type the following as your setup string:

```
\0270
```

\027 is the Escape, and the final 0 is the digit zero.

Newer printers, especially laser printers, can have elaborate setup strings, 15 or 20 characters in length.

■ *Page Limits and Other Settings*

Page limits refer to the size of your paper and the margins you want Javelin to leave (see Table 13.3 and Figure 13.22). The three toggle settings below the Limits settings refer to how Javelin should break up the pages.

The Paper Saver setting is useful only when you are printing a worksheet that is too wide to fit on the page with the margins you specify. Javelin handles wide worksheets by printing the worksheet in strips as wide as your paper. Each strip starts on a new page, so that you can tape them together.

If your worksheet is very wide but only a few rows long, the Paper Saver setting can be set On to tell Javelin not to skip to a new sheet of paper before beginning each strip. The Paper Saver setting applies only to printing worksheets, since no other view is wider than a page. (The Diagram view looks wide, but each diagram shows the connections for one variable, and the screens do not connect for printing purposes. There is no way to get Javelin to print one big diagram of your whole model.)

■ *Headers and Footers*

Title information in the body of a worksheet has the advantage that it can take up as many lines of text as you need, but it appears only on the first page of the worksheet when you print it. *Headers* and *footers* are lines of text printed at the top and bottom, respectively, of each page. You can include a title, your name, and other information in them.

You can also use these special commands in headers and footers:

- @DATE: Prints today's date.
- @TIME: Prints the current time.

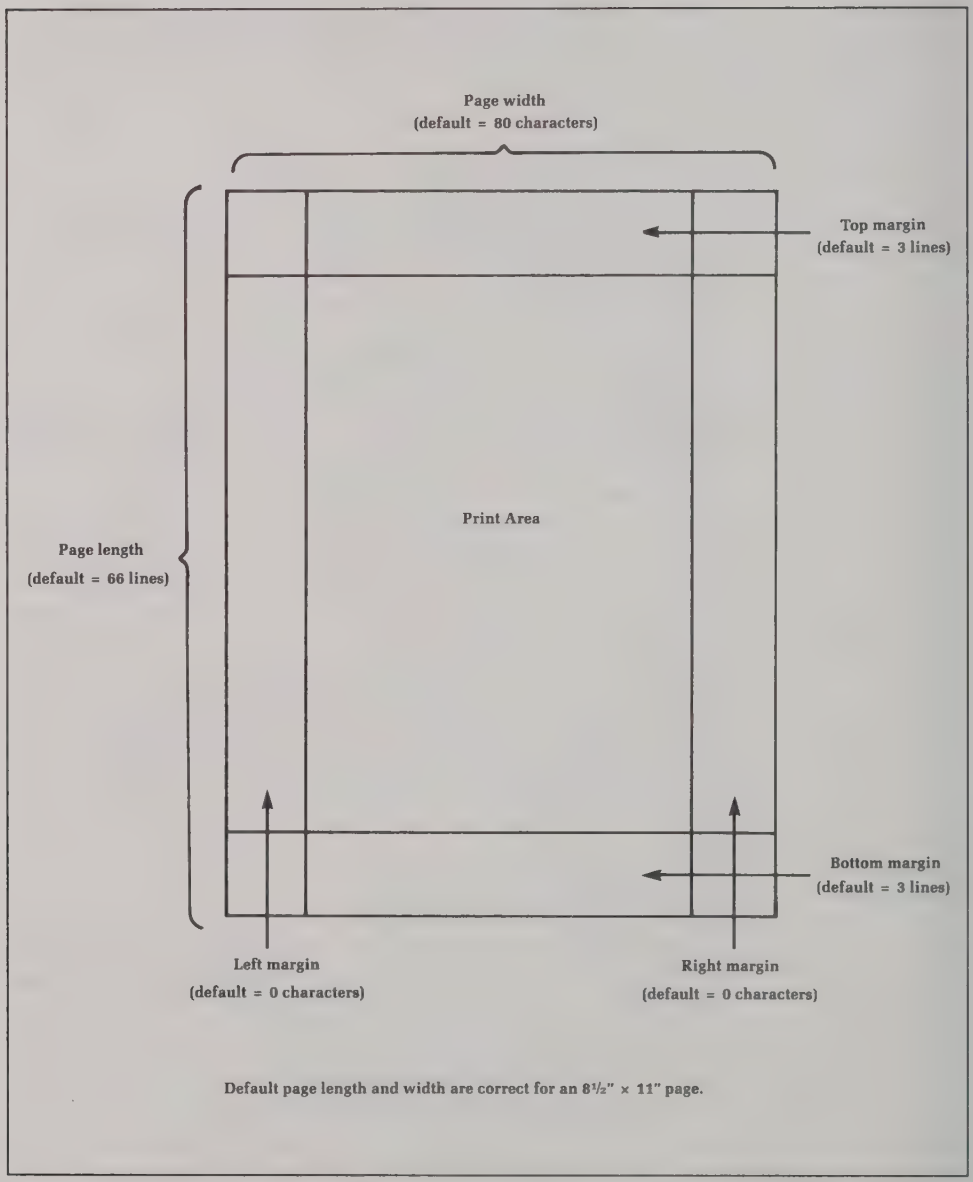


Figure 13.22: Page limits and margins

- @PAGE: Prints a page number, starting with 1 for each printout.
- @(*name*): (Javelin PLUS) Prints the contents of the variable name, which must be a constant containing a text (string) value. Be sure to enclose the variable name in parentheses.

Javelin gets the current date and time from your PC's system clock. If you do not have a clock/calendar to set this automatically, make sure you use the DOS DATE and TIME commands to set the clock correctly before running Javelin.

You can also control the spacing across the page of headers and footers by using vertical bars, which on most PCs are typed by Shift-Backslash. Javelin considers a header or footer to consist of three parts, separated by vertical bars:

Left-aligned text | Centered text | Right-aligned text

When the header or footer is printed out, Javelin replaces the vertical bars with the appropriate number of spaces. For example, the header

Page @PAGE | Budget Report | As of @DATE

will be printed as

Page 1 Budget Report As of 12/1/87

but with the actual date. These headers and footers will be printed when you print out any view from the model.

In Javelin 1.1 and later, you can also specify headers and footers for each worksheet, by entering information on the Worksheet settings sheet. There is space for up to five lines of header information and one footer line. If you enter headers and footers in both the Printer and Worksheet settings sheets, both sets will print, first the header from the Printer settings sheet, then the header from the Worksheet settings sheet.

The special functions @DATE, @TIME, and @(name), as well as the vertical bar, work only in headers and footers. In Javelin PLUS, you can enter the special function @PAGE in a worksheet to tell it to start a new page when printing.

■ Saving Printer Settings

Once you have entered information into the Printer settings sheet, you can save the settings permanently. These settings are stored once for Javelin itself, not separately with each model.

The printer settings are stored on disk in the same file (JAV.CFG) as the configuration settings from the Configuration settings sheet. The command to do this is located on the Configuration settings sheet. Display the Configuration settings sheet using / **D**efine **C**onfiguration. Move your cursor to the setting

Save configuration settings permanently

and press ⟵. This saves both the configuration and the printer settings. If you have saved the settings before, this new JAV.CFG file replaces the previous one.

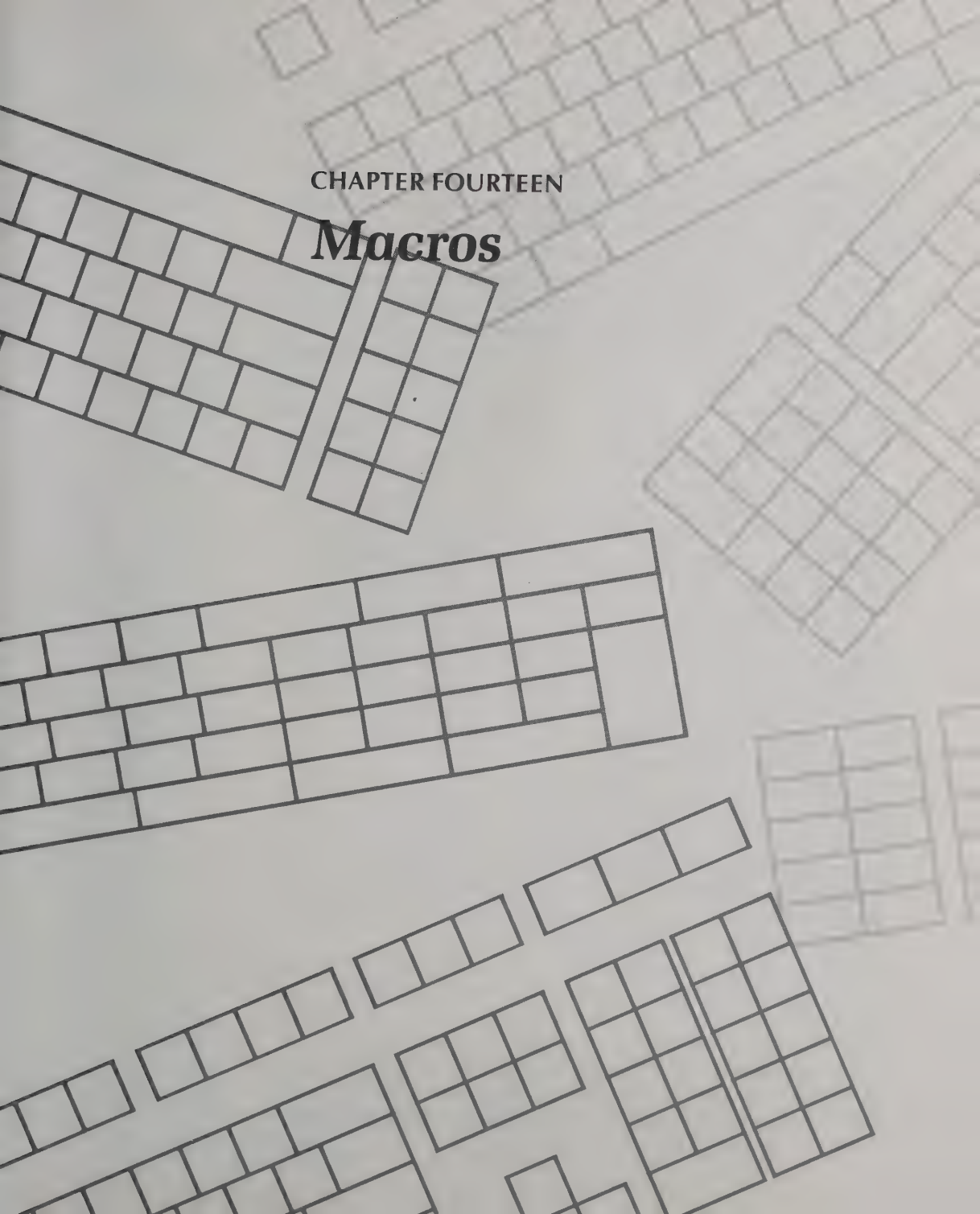
The header and footer information on the Worksheet settings sheet is saved with the model. In Javelin PLUS there is a Save Settings setting on the Printer settings sheet, so you can save your printer settings there.

Conclusion

■ Use Javelin's powerful Worksheet view to develop your model as well as to present data in tabular form exactly as you want it. The design of your model, particularly the way you structure your variable names, enables you to use indented left borders and superheadings in the top border. By combining multiple borders and the / *Worksheet **N**ame change command, you can create parallel sets of variables as you did for the Budget, Actual, and Variance figures.

Worksheet range calculations break the mold of the way Javelin formulas usually work. Experienced spreadsheet users will appreciate the fact that they allow you to tell Javelin that the graphic representation of your data on the worksheet (such as the relationship between a total and its detail lines) is in fact a part of the structure of your model.

We use worksheets even more often for creating and manipulating the model than for setting up reports, because of the wide range of shortcuts that they provide. Be sure to create as many worksheets in your models as you need. They give Javelin its power as a data presentation tool as well as a data analysis tool.



CHAPTER FOURTEEN

Macros

When using Javelin, you may find yourself repeating a series of commands over and over. You can save these commands a *macro,* a series of Javelin keystrokes and commands that are stored for future use. Such a series of keystrokes has a name and is stored as part of a model. You can perform the series of commands by pressing F7, the Macro key, and entering the name of the macro. Think of a macro as a piano roll for a player piano; when you play the macro, "magic fingers" type the stored keystrokes on your keyboard.

There are three situations in which you should create macros. You may want them for use right away, and use them only two or three times before discarding them: these are known as *convenience macros.* You may want to create macros that help you in your own work and that you use repeatedly: *personal macros.* You can also use Javelin's macro facilities to create complete *applications* for other users.

A convenience macro is handy when you find yourself doing a great deal of repetitive typing. For instance, you want to lock many of the variables in your model. While locking the first one, record this as a macro. Javelin can then replay the keystrokes to lock each additional variable, one at a time.

A personal macro you might want to include in all your models could be one you run whenever you are done working on a particular model. It could save the model, print a worksheet, print a Formulas view, and clear the model. Personal macros can also help you automate calculations that you cannot write completely in Javelin formulas, for example, recalculating a circular model until the answer stabilizes. You can write these macros once and then import them into all your models.

Neither convenience macros nor personal macros call for real programming ability; if you know how to use Javelin, you can create them. Applications, on the other hand, create systems complete enough that the person using the application does not have to know Javelin and may even be unaware that the application is written in Javelin. This does require programming ability, but as you create simple Javelin macros, ultimately you are learning to write more complicated ones.

Macros consist mostly of keystrokes to be "typed," but there are also special *macro commands* that can be used only as part of a macro. Macro commands let you put tests and loops into macros, create your own menus, and make a macro pause so that you can enter input while it is running.

In this chapter we will look at three models. In the first one, you will use saved keystrokes to help create the logic of a large, repetitive model that creates a histogram of a data series. This is an example of a convenience macro.

In the second, you will use conditional macro commands and looping to automate the calculation of the circular Compensating Balances model introduced in Chapter 11. If you encounter such a situation often, you might want to consider making this a personal macro. In the third model, you create a self-contained Javelin mortgage application that requests some input from the user and prints reports and graphs for payment and amortization tables.

You'll use the commands / **D**efine **M**acro and / **V**iew **M**acro and work in the last of the ten Javelin views you've seen—the Macro view.

Macro Basics

■ No matter what a macro is used for, there are two ways to begin it. Either you tell Javelin to record your keystrokes while you type the series of commands, using the / **D**efine **M**acro commands to tell Javelin when to begin and end recording the macro. Or you edit macros directly, using the / **V**iew **M**acro command to display a Macro view. Once you have created a macro with / **D**efine **M**acro, you can view and edit it in the Macro view. The Macro view looks very much like the Notes view, and you edit macros in the same way you edit notes.

Let's record a macro that will change the prefix for each label in a column of cells on a worksheet. If you enter a column of labels and then decide that you want them centered, you have to edit each one to change the label prefix from an apostrophe to a caret. But you can record the series of keystrokes required to change one label prefix and then play it for each label in the column, which is much easier than doing it all by hand. (Javelin PLUS has a command that does this—/ *Worksheet **J**ustify.)

Start with an empty model, display a worksheet, and type the following labels into the left border:

One
Two
Three
Four
Five
Six
Seven

They are automatically prefixed with an apostrophe, which positions the labels at the left side of the cell.

■ Recording a Macro

Now you're ready to record the set of steps necessary to change the label prefix for one cell from an apostrophe to a caret. Move the cursor to the first label cell. Tell Javelin to start recording the macro, giving the command / **D**efine **M**acro Begin. Enter the name of the macro:

```
Center
```

The RECORD indicator appears on the status line, showing that Javelin is recording every keystroke. To change the label prefix, edit the label (the edit cue appears at the beginning of the label on the edit line), delete the existing label prefix, type the new one, and move down to the next row. Type the keystrokes to do this: press F2 and Del, type a caret, and press ↓. Now tell Javelin to stop recording, with the command / **D**efine **M**acro End. Javelin confirms that the macro has been defined.

If you tell Javelin to record a macro that already exists, it asks whether you want to replace the old definition of the macro, add new material to the end of the macro, or cancel the command.

■ Running a Macro

To run a macro, press the Macro key and tell Javelin the name. In this case, press F7, type

```
Center
```

and press ⟵. Javelin performs the same steps you just did, only much faster.

Once you have longer macros, you may need to cancel a macro while it is running (especially if it seems to be running amok). You can do so by pressing Ctrl-Scroll Lock (also known as *Break*).

One-Letter Macro Names Macro names can be very long and contain spaces as well as other special characters. But there is a reason to keep some macro names short: if you give your macro a one-letter name, such as Z, you can run it by pressing the Alt key and the letter—in this case Alt-Z. This is useful if you are building a convenience macro for immediate

use or if you include a few standard functions in all your models (such as P for recalculate and print or S for save). But if you have many macros in a model, don't give them all one-letter names: in six months, you won't remember what macros W, R, and U do.

To rename a macro, use the command / **D**efine **M**acro Rename, then type in the existing name and the new name. Rename the Center macro as C now. Run the macro by pressing Alt-C, and Javelin centers the next label. You can center all the labels by pressing Alt-C repeatedly.

The Macro View

■ In the Macro view, you can look at and change the contents of a macro, that is, the actual commands that make up the macro. View the one you just wrote by giving the command / **V**iew **M**acro C (see Figure 14.1). The keys you can use while working in the Macro view are the same as those you use in the Notes view (see Table 14.1).

■ Macro Commands for Keystrokes

Most of what you will see in the Macro view are the macro command equivalents of keys you press all the time. Printing characters such as letters, digits, and punctuation appear as themselves. Other keystrokes appear as macro commands, which are words that begin with an "at" sign. There are macro commands for cursor keys and other keys that do not print. Table 14.2 shows the macro commands for the cursor movement keys, function keys, and other keys.

There are macro commands for three printing characters: @SPACE, @ENTER, and @SLASH. When Javelin runs a macro, it disregards spaces and line breaks in the macro. This allows you to format your macro to make it easier to read, without changing how it runs. Use @SPACE to tell it that you

```
Macro C:

@F2 @DEL ^ @ENTER
@DOWN
```

Figure 14.1: Macro view of macro C

really want a space and @ENTER to say that you really want ⟵. There is also a macro command @SLASH because, when you are in the Macro view, if you type an actual slash Javelin brings up the Main menu. Use @SLASH in your macro to make it "type" a slash when you run the macro. Use @@ if you need to include an at sign in the macro.

The Macro view lets you change macros you have defined without retyping them from scratch. For example, if you decide that you want your labels aligned to the right, you can edit the macro in the Macro view, changing the caret to a double quote.

Debugging and Explaining Macros

■ When you create a macro, either by recording it or by typing it in the Macro view, you should test it to make sure that it does what you want.

Cursor Movement:

@LEFT	@RIGHT	@UP	@DOWN
@HOME	@END	@PGUP	@PGDN
@CTRLLEFT	@CTRLRIGHT		
@CTRLHOME	@CTRLEND		

Function Keys:

@F1	@F2	@F3	@F4
@F5	@F6	@F7	@F8
@F9	@F10	@ALTF3*	@ALTF6*

Other Keys:

@SPACE	@ENTER	@SLASH	@ESC
@INS	@DEL	@BKSP	@SPELL
@CTRLQ	@@	CTRLBKSP*	

*Javelin PLUS only

Table 14.1: Keys in the Macro view

■ *Single-Stepping Execution*

The most reliable way to test a macro is to tell Javelin to run it in slow motion. This is called *single-stepping* a macro, because it does just one step at a time, pausing after each step so you can see what has happened. Otherwise, macros run so quickly that it is hard to tell whether they are running correctly.

To single-step a macro, press Alt-F7 to run it. The STEP indicator appears on the status line while Javelin is running the macro in single-step mode. Javelin pauses after each keystroke in the macro.

Key Name	Function
←	Moves the cue one character to the left.
→	Moves the cue one character to the right.
↑	Moves the cue one line up.
↓	Moves the cue one line down.
Home	Moves the cue to the beginning of the current line.
End	Moves the cue to the end of the current line.
PgUp	Moves the cue up one screenful of lines.
PgDn	Moves the cue down one screenful of lines.
Ctrl-Home	Moves the cue to the beginning of the macro.
Ctrl-End	Moves the cue to the end of the macro.
Del	Deletes the character the cue is on.
⟵	Deletes the character immediately to the left of the cue.
↵	Types a new-line character, breaking the line the cue is on into two lines.
⟶|	The Spell key here helps you type macro commands such as @ENTER.
Ctrl-Q	Quotes the next character, that is, takes the next character literally, rather than interpreting it as a command. For example, to type a slash into a macro, type Ctrl-Q followed by /. The slash appears in the macro, rather than telling Javelin to display the menu.

Table 14.2: Macro commands for keystrokes

You can press the following keys to tell Javelin how to continue:

- Space: Continue single-stepping the macro.
- Esc: Cancel the rest of the macro.
- F7: Continue running the macro at full speed.

Try running the C macro in single-step mode. Press Alt-F7, C, and ⟵. Press a space each time the macro pauses to tell it to continue.

Type a column of numbers next to the column of labels and try running the C macro with the cursor on a cell containing a number. You'll see that the macro deletes the first digit of the number and replaces it with a caret, changing the rest of the number to a label.

So watch out where your cursor is when you run a macro. Keystrokes that do useful work with your cursor in one view or cell may destroy information in another view or cell. This is perfectly acceptable for a convenience macro because you are going to create it, use it, and discard it. If you decide that you want to keep this macro and use it in many models, or if you want to give it to your associates, use the techniques described below to make it more useful and less dangerous.

■ *Documenting Macros*

Documenting a model is important; documenting macros is doubly important. Macros are harder to read than formulas, and an unstructured and undocumented macro is usually incomprehensible.

You document your macro implicitly by laying it out neatly, and explicitly by adding comments. We have laid out the macros in this chapter in a simple way so that each command is on a separate line and is indented to show the macro's structure. Arranging the macros this way took a few extra minutes, but the time spent is amply repaid whenever we have to go back to understand or change the macros.

You should also put comments into your macros to explain what they do, using @REM to introduce comment lines. Javelin ignores everything on the line after the @REM.

```
@REM This macro prints all worksheets.
@ REM Written by J. Levine, 6/87.
```

Particularly long or clever macros always benefit from a description at the top of what they do.

Interactive Macros

■ To make your macros more generally useful, you can have them pause, ask for information, and continue after you have typed the information. Macros that work this way are called *interactive macros* because they interact with the user as they run.

There are four interactive macro commands:

- @PROMPT *text*: Displays the text on the information line.
- @READ: Pauses to accept a single line from the keyboard. As soon as you press ⟵, the macro continues after the @READ command. Javelin has no way of knowing what was typed by the macro and what was typed by the user in response to a @READ command. This is useful when you select variable names or enter single pieces of information. The ⟵ that the user types to end the entry is not typed as part of the macro; it just tells Javelin that typing is finished. If you want a ⟵ at that point, you must put an @ENTER in your macro.
- @PAUSE: Pauses, putting the macro on hold. While the macro is on hold, the PAUSE indicator is displayed on the status line and you can enter information or move the cursor around the current view. The slash and function keys are disabled, so you cannot give commands. To continue running the macro, press F7.
- @BEEP: Beeps to alert the user that something is happening.

■ Accepting a Single Piece of Information

If you want to let the user type in a piece of information in the middle of a macro, such as a variable name, date, number, or file name, first display a prompt using @PROMPT, and then use @READ to accept the information. When the user presses ⟵ after typing the information, what is typed becomes part of the macro.

For example, here is a macro that sets the default column width for a worksheet to five characters wide:

```
@SLASH D W
@PROMPT Enter the name of the new worksheet:
```

```
@READ                    @ENTER
@DOWN                    5 @ENTER
@ESC
```

This is what the macro does:

- @SLASH D W gives the command / **D**efine **W**orksheet. At this point, Javelin is waiting for the name of the worksheet.
- @PROMPT displays the prompt on the information line to encourage the user to type the name. Javelin is still waiting for a worksheet name.
- @READ accepts what the user types, not including the ending ⟵, which just signals that typing the information is complete. The information that the user typed is used as the worksheet name Javelin was waiting for.
- @ENTER presses ⟵ after the worksheet name so that Javelin knows that the name is complete. Now Javelin displays the Worksheet settings sheet. The cursor is on the first item of the settings sheet, Name.
- @DOWN moves the cursor down to the next item, Column Width.
- 5 @ENTER types in the new column width and presses ⟵.
- @ESC exits from the Worksheet settings sheet.

■ *Pausing for Full-Screen Data Entry*

In other situations, you would like the macro to pause and let the user type as much information as desired into the view that is displayed. To do this, use the @PROMPT command to suggest what information should be entered and then use the @PAUSE command. While the macro pauses, the user can type in information and move around the current view. But Javelin ignores any attempt to give a command or press any function key except F7, which continues the macro.

Although you can use @PAUSE in any view, the most useful are the Table, Chart, and Worksheet views, so that the user can type in a list of data; for example, a year of monthly stock prices in an analysis model.

Variable Information in Macros

■ There are times when you want a macro to remember some aspect of what it is doing. For example, you may want the user to enter part of a variable name and then create three or four variables using that name. Clearly, you do not want to make the user enter the name four times. Or you may want to repeat a process a number of times (see the histogram example below). Or you may want to have Javelin invert the word order in some labels on the side of a worksheet. In all these cases, Javelin should pick up a piece of information from one part of your model and use it in another.

Javelin's macro facility does not have variables; there is no direct way for you to tell Javelin in a macro to type the contents of a variable. But you can accomplish the same thing by using the Ctrl-K and Ctrl-Y key combinations to "cut and paste" information.

- Ctrl-K: "Kills" (deletes) the text that follows the edit cue on the edit line, or on the current line in the Macro or Notes view.
- Ctrl-Y: "Yanks" (inserts) the text that was most recently "killed," inserting it where the edit cue is.

In effect, Ctrl-K and Ctrl-Y give you one "variable" (called the *kill buffer*) that you can use in macros. This allows you to move information among macros, notes, formulas, worksheet labels, and data values. The text most recently deleted with Ctrl-K is placed in the kill buffer. Each time you press Ctrl-Y, Javelin types the contents of the kill buffer wherever the cursor is.

Although you can use Ctrl-K and Ctrl-Y at any time, they are particularly useful in macros. The macro commands to do these operations are @CTRLK and @CTRLY.

Here's a simple example. You want to let the user enter a variable name (X, for instance). For the name to be entered, you want to create three variables: X INPUT, X USE, and X ESTIMATE. X INPUT will be input data, X ESTIMATE applies 10-percent growth to the previous X USE, and X USE is X INPUT, if there is any, or X ESTIMATE otherwise. The formulas you need

would be:

```
X INPUT = <Monthly Data>
X ESTIMATE = PREVIOUS(X USE) * 1.1
X USE = X INPUT; X ESTIMATE
```

Notice that X appears six times in these formulas. You can have the user type X six times, or you can use Ctrl-K and Ctrl-Y. A macro to do this might be as follows.

```
@REM Let the user type in the variable name:
@PROMPT Enter the variable name:
@READ

@REM Put it in the kill buffer:
@HOME @CTRLK

@REM Define the USE variable as the INPUT or GROW variable:
@CTRLY @SPACE USE @SPACE = @SPACE @CTRLY @SPACE
  Input;
@SPACE @CTRLY @SPACE GROW @ENTER

@REM Define the GROW variable in terms of the USE variable:
@CTRLY @SPACE GROW @SPACE = @SPACE PREVIOUS
  ( @CTRLY USE )
@SPACE * @SPACE 1.1 @ENTER

@REM And put the user in the Table view to enter the INPUT data:
@SLASH vt @CTRLY @SPACE INPUT @ENTER
```

The Histogram Model

■ Here is a useful model to create *histograms*, which also demonstrates use of Ctrl-K and Ctrl-Y. A histogram is a chart that counts the number of values in a series by range. For example, if you have a series of sales data with values ranging from 0 to 100, you might want to know how many values are in the range 0 to 10, how many from 10 to 20, and so forth. Javelin

makes it straightforward to build a histogram, but it involves a lot of repetitive typing. Fortunately, you can use a macro to avoid most of the typing. (1-2-3 has a command, Data Distribution, to make histograms.)

Rather than typing a long series of data to analyze in your histogram, let's make one up with a formula. The variable R in the model in Figure 14.2 is a year-long series of random numbers ranging from 0 to 1000. If the numbers are truly random and you divide up the range from 0 to 1000 into ten sections, there should be roughly the same number of values in each section. This model will test how random these numbers are.

The variables H 1 through H 10 are constants (remember, TSUM returns a constant) that count the number of values in each section. In variable H 1, for example, the expression

AND(R >= 0, R < 10)

will equal 1 for each day on which R is in the range 0 to 10, and 0 otherwise (AND returns 1 if both its arguments are true, 0 otherwise). The TSUM function adds up all the ones in the year to give the count of the number of values in the range 0 to less than 10. Variables H 2 through H 10 count the number of variables in the other nine ranges the same way.

Javelin has no good way to graph ten constants against each other. But if you make them into a time series called Hist with the value of H 1 at date 1 A, H 2 at date 2 A, and so forth up to H 10 at 10 A, you can create a graph of Hist for periods 1 A through 10 A.

```
Formulas for all variables:

R = DAILY(INT(RANDOM() * 1000))
Count = 1 + SINCE(1 A)
Range = 100 * SINCE(1 A)

H 1 = TSUM(AND(R >= 0, R < 100))
H 2 = TSUM(AND(R >= 100, R < 200))
H 3 = TSUM(AND(R >= 200, R < 300))
H 4 = TSUM(AND(R >= 300, R < 400))
H 5 = TSUM(AND(R >= 400, R < 500))
H 6 = TSUM(AND(R >= 500, R < 600))
H 7 = TSUM(AND(R >= 600, R < 700))
H 8 = TSUM(AND(R >= 700, R < 800))
H 9 = TSUM(AND(R >= 800, R < 900))
H 10 = TSUM(R >= 900)
Hist = H 1 @ 1 A;H 2 @ 2 A;H 3 @ 3 A;H 4 @ 4 A;H 5 @ 5 A;
     H 6 @ 6 A;H 7 @ 7 A;H 8 @ 8 A;H 9 @ 9 A;H 10 @ 10 A
```

Figure 14.2: Formulas for Histogram model

You could type this in by hand, but it would be tedious. Instead, define a macro that writes the formulas for your H variables. The formulas for the H variables differ in the number part of the name, the bottom of the range, and the top of the range. The easiest way to make the macro handle these variations is to set up a worksheet that has the necessary parts in parallel columns on the worksheet, so that the macro can use Ctrl-K and Ctrl-Y to pick up the parts and put them into the formulas.

Figure 14.3 shows a worksheet ready to create the histogram formulas. The border has the custom dates from 1 A to 10 A. The first column has the numbers from 1 to 10 to use in the variable names, and the second column has the numbers from 0 to 1000 to use for the range limits.

Let's look at the first significant macro in Figure 14.4. It takes the count and range values on the worksheet and turns them into a variable such as H 1, which is entered as an exception cell in the third column. We created this macro by recording our keystrokes as we created the first histogram variable, then going into the Macro view to clean up and comment on it.

Start in the first row of the first column of the worksheet, with the value 1, the name counter for the first H variable. The first line of the macro uses @F2, the Edit key, to copy the contents of the cell to the edit line, saves it to the kill buffer with @CTRLK, and then presses @ESC to abandon the edit so the cell is not changed. It then moves two cells to the right to the cell where the new variable is to be created.

The next two lines create the variable, but use place holders *xx* and *yy* where the upper and lower limits will be. The variable now looks like this:

```
H 1 = tsum(and(r >= xx, r < yy))
```

@CTRLY puts the saved number into the variable name, and @SPACE puts spaces into the formula.

			Hist
1	A	1	0
2	A	2	100
3	A	3	200
4	A	4	300
5	A	5	400
6	A	6	500
7	A	7	600
8	A	8	700
9	A	9	800
10	A	10	900

Figure 14.3: Worksheet ready to create histogram formulas

The next line uses @CTRLK to get the lower limit from the cell to the left of the one with the formula. Then you substitute the lower limit for the dummy xx. If you press F5 (Goto) while editing something on the edit line, Javelin prompts you for a string to search for in the edited material. (This works only in Javelin 1.1 or later.) Use this facility to find xx, then push @BKSP twice to backspace over the xx, use @CTRLY to get the saved lower range value, @BKSP again to backspace over the space that always follows a number killed from the edit line, then reenter the modified formula, which now looks like this:

```
H 1 = tsum(and(r >= 0, r < yy))
```

The next line picks up the upper limit for the range from the cell to the left and below the formula's cell, and the two following lines substitute the upper limit for the place holder yy. The last line of the macro moves back to the first column and down to the next row so you are ready to run the macro on the next row.

Having defined the macro H while creating the first variable, you can then run it several times by pressing Alt-H, each time creating the next variable. For the last variable, H 10, the macro will fail because there is no value for the upper limit. In this case the macro stops with the formula still on the edit line, and it is easiest to adjust the last formula by hand.

```
Macro H:

@REM Pick up the number of the formula.
   @F2 @CTRLK @ESC @RIGHT @RIGHT

@REM Enter the formula with dummy limits.
   H @SPACE @CTRLY @SPACE = @SPACE tsum(and(r @SPACE >= @SPACE xx,
   @SPACE r @SPACE < @SPACE yy)) @ENTER

@REM Pick up the lower limit.
   @LEFT @F2 @CTRLK @ESC @RIGHT

@REM Substitute it into the formula for xx.
   @F2 @F5 xx @ENTER @BKSP @BKSP @CTRLY @BKSP @ENTER

@REM Pick up the upper limit from the next line.
   @LEFT @DOWN @F2 @CTRLK @ESC @RIGHT @UP

@REM Substitute it into the formula for yy.
   @F2 @F5 yy @ENTER @BKSP @BKSP @CTRLY @BKSP @ENTER

@REM Move to the beginning of the next line.
   @DOWN @LEFT @LEFT
```

Figure 14.4: Macro to create a single histogram range variable

Having created the ten separate histogram variables, now you want to glue them together into one time series that you can graph. Naturally, a macro would make this easier. The new macro is actually simpler than the previous one. This time, make sure the screen is split into two windows, and put the worksheet in the top window and the Formulas view in the bottom. Then create the first version of variable Hist in the Formulas view:

```
Hist = H 1 @ 1 A
```

The next macro, G, will add one of the H variables to Hist. In the Formulas view, the cursor should be on Hist, as in Figure 14.5. Then switch to the other window and put the cursor on the cell containing 2, since the next variable to add to Hist will be H 2 @ 2 A. These positions are important so that the cursor will move to the right place when you use @F6 in the macro (see Figure 14.6).

In the first line you pick up the variable number with @CTRLK. Then switch to the other window and use @F2 to edit the definition of Hist. Next, go to the end of the definition and add the reference to the new variable. Notice that @CTRLY appears twice, once to enter the number as part of the variable name (e.g., H 2) and once to enter it as part of the custom date (e.g., 2 A). The last line switches back to the worksheet and moves down to the next line, ready for the next run of the macro.

Having recorded this macro, you press Alt-G eight more times to add variables H 3 to H 10 to the definition of Hist.

Figure 14.7 is a bar graph of Hist, which reveals that the number of

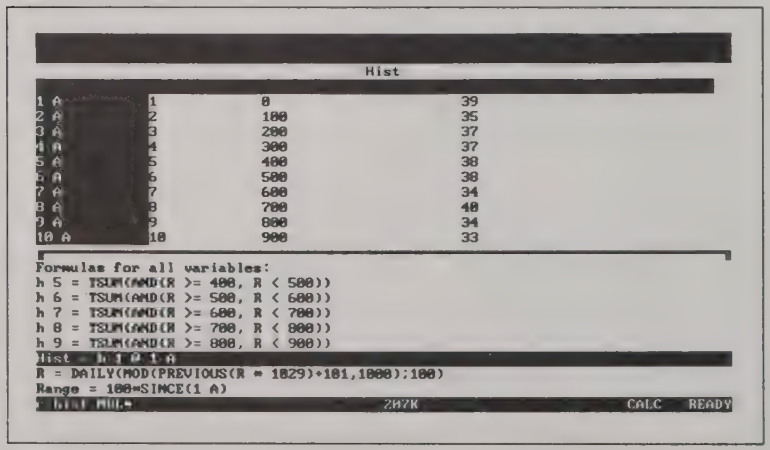


Figure 14.5: Preparing to add a variable to Hist

```
Macro G:

@REM Pick up the formula number (also time period number).
   @F2 @CTRLK @ESC

@REM Switch to the other view, edit Hist.
   @F6 @F2

@REM Move to the end, add ;h <n> @ <n> A.
   @END ;h @SPACE @CTRLY @@ @SPACE @CTRLY A @ENTER

@REM Switch back to first view, move to next number.
   @F6 @DOWN
```

Figure 14.6: Macro G which adds a variable to Hist

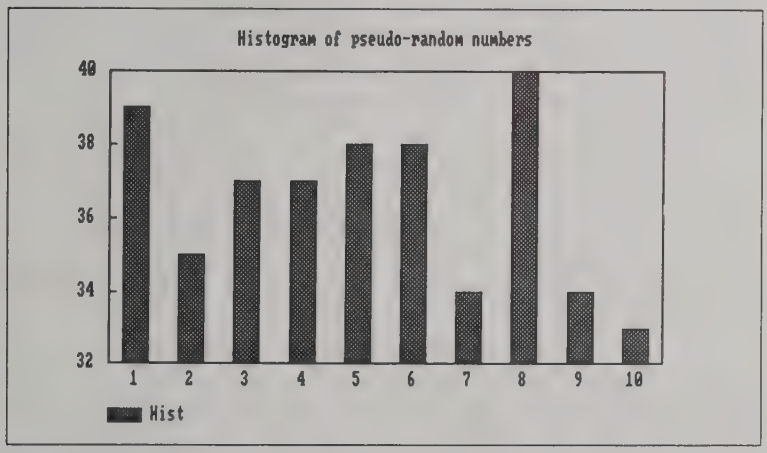


Figure 14.7: Graph of completed histogram

values in each section ranges from 33 to 40—not bad for random numbers. Your numbers will be different, since Javelin's random-number generator gives you different random numbers each time.

Conditional Processing

■ When working on a model, you frequently look at the values and make a decision based on what you see. Javelin's macro facility can do this as well.

This is called *conditional processing,* or macro execution that depends on a condition. For example, your macros can ask you a question using @PROMPT and @READ and proceed according to your answer, or they can do different things depending on the value of a variable in the model.

The macro commands for conditional processing are:

- @IF *condition:* Begins the conditional statement, and tells Javelin what the condition is.
- @THEN *commands:* If the preceding @IF condition is true, the commands following @THEN will be executed. @THEN must follow each @IF condition.
- @ELSE *commands:* If the @IF condition is false, the commands following @ELSE will be executed. (If you don't want the macro to do anything if the condition is false, leave out @ELSE.)
- @ENDIF: Ends the @IF statement.

You can test three kinds of conditions:

- Is a constant true? For example, @IF Budget Exceeded.
- Is a time-series variable true at a particular date? For example, @IF Total Budget @ 1988.
- Has there been an error? Errors occur in macros the same way they occur when you are using the program normally: you (or your macro) type something that Javelin can't understand. Test for them using @IF @ERROR.

When testing the value of a variable or a variable at a date, Javelin considers the condition to be true if the variable's value is not zero. Note that macros do not evaluate any formulas. That is, if you want to test Budget Total > Budget Limit, you must create a constant or time-series variable that has either true or false values, and then test that. You could create a variable Budget Exceeded, like this:

Budget Exceeded = (Budget Total > Budget Limit)

You can then use Budget Exceeded in your @IF command.

You are probably familiar with the low-pitched beep that Javelin makes when it finds something it can't do. Your macro can listen for the beep by

using the @ERROR macro command. @IF @ERROR tests to see if an error has occurred since the macro started running (or, in version 1.0 only, if an error occurred even before the macro started). Each subsequent @IF @ERROR after the first checks to see if there has been an error since the last @IF @ERROR.

Starting with Javelin 1.1, you can also use @ERROR to create an error condition. Unless you are a serious programmer, you won't need this feature.

■ Controlling the Order of Execution

When doing conditional processing, you can tell Javelin to execute different parts of the macro depending on the condition, or to execute a part more than once. For example, you could calculate a circular model over and over until the values calculated converge to stable numbers.

Here are the macro commands that let you control the order in which the macro is executed:

- @LABEL *label*: Assigns a name to this place in the macro. A label may not contain any spaces.
- @JUMP *label:* Tells Javelin to jump to the label specified to continue executing the macro there.
- @EXIT: Exits from the macro.

Consider the Compensating Balances model from Chapter 11. This model requires that you push F9 (Calc) repeatedly until the model converges. Figure 14.8 shows a macro that uses @IF and @JUMP to recalculate the Compensating Balances model. It pushes @F9 to recalculate the model, then tests the variable Converged to see if the values are consistent. If so, it uses @EXIT to leave the macro, otherwise it @JUMPs back to recalculate again. Figure 14.9 shows the Compensating Balances model with the addition of the variable Converged.

```
Macro Run:

@LABEL loop
    @F9     @REM Calculate the value of the model.
    @IF Converged @THEN @EXIT @ENDIF
    @JUMP loop    @REM Not done, do it again.
```

Figure 14.8: Macro to recalculate a circular model

You want to test two things, that Compensating Balance is indeed 10 percent of the loan, and that Total Loan is indeed the sum of Cash Loan and Compensating Balance. Numeric round-off error can make these values never quite agree, so check to see if the absolute value (ABS) of the difference is less than .01, which means that they agree at least to the cent.

■ Running One Macro From Within Another

If you write a number of macros, you may find that in the middle of running one macro, you want to perform steps that you have already recorded in another macro. For example, you may have written a macro SavPrn that saves your model and prints out a worksheet. While writing a second macro called Update that makes some changes to the model, you realize that Update should also save the model and print the worksheet. Rather than repeating the commands in Update, you can run SavPrn from within Update.

The Update macro is said to *call* the SavPrn macro, and the SavPrn macro is called a *subroutine.* Calling one macro from within another is also called *nesting macros.*

To run one macro from within another, you use the macro command @F7. Just as when F7 is used from the keyboard, this runs a macro. But since Javelin is already running a macro, it stops running the first macro and runs the second. When the second macro has run, Javelin returns to the first and continues executing it, starting right after the @F7 command.

You can nest more than one macro at a time. In Javelin 1.0, you can call up to five macros deep, one from within the next. In Javelin 1.1 that limit was raised to ten. A macro can even call itself, although that is not usually a good idea; it tends to exceed the maximum nesting level very quickly. If you think your macro needs to call itself, you should probably be using a @JUMP command to loop within the macro. You can use @F7 only in a macro when you could have used it from the keyboard—that is, not inside menus, settings sheets, building blocks, and the like.

```
Formulas for all variables:

Cash Loan = 15000
Compensating Balance = 0.1 * Total Loan
Total Loan = Cash Loan + Compensating Balance
Converged = AND(ABS(Compensating Balance - 0.1 * Total Loan) < 0.01,
    ABS(Total Loan - (Cash Loan + Compensating Balance)) < 0.01)
```

Figure 14.9: Compensating Balances model with convergence test

■ More Automation in the Histogram Model

In the Histogram model above, you had to press Alt-H for each H variable created. That is not too bad for a 10-element histogram, but it would be very inconvenient if your histogram had 100 elements. Figure 14.10 shows a way to make the process more automatic.

Above, when you got to the last of the list of numbers on the worksheet, the @F2 (Edit) function failed and Javelin beeped. You can use the @ERROR test to check when this happens, so you know when you reach the end of the list. The new macro HLoop uses @F7 to call the macro H and checks to see if an error occurs. If it has, it exits. If not, it jumps back to call H again. Once you have set up H, you can use HLoop to run it automatically, so that you can do something else while your computer runs the macros.

In this case you could have put the test and jump into the H macro rather than creating a new HLoop macro. It is much better practice to create a second macro here. Had you edited the changes into the H macro, you might inadvertently have changed something else, put the label or the jump in the wrong place, or otherwise introduced errors into your formerly working macro. By creating a separate macro to do the looping, you can ensure that your existing macro H still works, and the only possible source of error is HLoop, which is much smaller and so easier to check.

It also turns out that jumping around in large macros is very slow, since Javelin has to scan the whole macro to find the label every time you make a jump. By keeping your macros small, you make them faster and more reliable.

■ Calling In Another Model

Starting in Javelin 1.1, macros can use / **F**ile **L**oad to load another model. (If you try it in Javelin 1.0, the results are highly unpredictable.) When you load a model, Javelin normally deletes the previous model, including all its macros.

```
Macro HLoop:

   @LABEL loop
@REM Create the next variable.
    @F7 H @ENTER
@REM If there was a problem we must be done.
    @IF @ERROR @THEN @EXIT @ENDIF
@REM Not done, do it again.
    @JUMP loop
```

Figure 14.10: Loop to create H variables automatically

There is a special case, though, for loading a model within a macro, also called *chaining* models together. If a macro issues a / **F**ile **L**oad command, that macro and only that macro is preserved when the new model is loaded. It continues to run but as soon as it stops, by reaching either @EXIT or the end, it is deleted. If that macro was called by nesting from other macros, the other macros are deleted and the nesting is canceled. Although the macro is kept in memory when the new model is loaded, it is in a sort of limbo since it no longer has a name and will be deleted automatically when done.

/**F**ile **L**oad allows you to chain models in two ways. One is to have a master model with a master macro. The master macro issues / **F**ile **L**oad commands to load other models and issues @F7 commands to run macros in those models. When these subsidiary macros finish running, control returns to the master macro, which can then load another model. Although this works, it means that your models run under the control of a macro that is in limbo. It can be hard to debug such a system.

Another way to chain models is to have the last macro in the first model load the second model and start a macro in the second model. When the second model is done, it can load a third model and run a macro in that model and so on to any level, and it is possible to return in the end to the first model by reloading it. Since the macro that loaded the second model immediately calls a macro that exists in the second model, if there are any problems, you can see the macro and fix it.

Macro Programming

■ So far, all the macro techniques we have looked at help Javelin to duplicate what you might do when using your model: type things in, give commands, look at the results, and do one thing in one case or another in a different case. Now we come to some commands that are only useful if you are programming applications in Javelin.

They allow you to set up your model so that all tasks you normally do appear on a menu of your own design. When you pick a task from your menu, Javelin runs the specified macro, so that the user never gives Javelin direct commands. You can also suppress Javelin's screen display, to keep the commands from flashing by on the screen and confusing the user. To write fancier macros like these, there are macro commands for displaying menus, controlling the appearance of the screen, and taking more control over how Javelin operates.

■ Macro Menus

Here are the macro commands you use to display your own menu:

- @SELECT *text:* Begins the macro menu. The text specified is displayed at the top of the menu and should tell the user what to do.
- @CHOICE *label text:* Specifies one choice on the menu. The text is displayed as the choice on the menu. If the user chooses that choice, Javelin jumps to the label and continues executing the macro. You may have up to twelve @CHOICEs in a menu.
- @ENDSELECT: Ends the macro menu.

When Javelin executes a macro menu, it displays the text specified in the @SELECT command and shows the list of choices in a query menu with the cursor on the first choice. The user moves the cursor up and down to highlight a choice and then presses ⟵. If you make each choice start with a different letter, the user can also choose by pressing the key representing the first letter of the desired choice. The macro menu works just like Javelin's built-in menus.

When the user has made a choice, Javelin jumps to the label for it and continues executing the macro there. You need no @JUMP command; the @CHOICE command tells Javelin to jump to the label. If the user does not make a choice (by pressing Esc), the macro continues at the command just after @ENDSELECT.

In the macro, it is a good idea to indent the @CHOICEs for the menu under the @SELECT command, so that it is obvious where the menu starts and ends.

■ Controlling the Screen

Normally, as a macro runs you can see each command flash by on the screen. If you don't want what the macro is doing to appear on the screen, use these commands:

- @SILENT: Tells Javelin to leave the screen as it is and not show what the macro is doing.
- @ENDSILENT: Tells Javelin to resume displaying what the macro is doing.

Other conditions also tell Javelin to resume the screen display:

- A @READ or @PAUSE command is encountered.
- The macro displays a Graph or Quick-graph view.
- The user cancels the macro by pressing Ctrl-Scroll Lock (Break).

@SILENT speeds up macros considerably, since otherwise Javelin has to spend time updating the screen. (In some copies of Javelin 1.1, @SILENT causes macros to behave unpredictably. If you have this problem, call Javelin Software.

■ *Controlling the Keyboard*

If you write macros to be run by other Javelin users, you may not want them to be able to cancel running the macros. You can program Ctrl-Scroll Lock's ability to cancel a macro.

- @BREAKOFF: Disables the Ctrl-Scroll Lock key combination.
- @BREAKON: Enables the Ctrl-Scroll Lock key combination.

Before you include @BREAKOFF into your macro, be sure that it does what you want. If your macro does a @BREAKOFF and then gets into a loop or starts printing an unwanted thousand-page report, there is no way to stop it short of restarting your computer entirely.

■ *The Auto-Executing Macro*

You can tell Javelin to execute a macro automatically when a model is loaded. All you have to do is to name the macro AUTOEXEC. When Javelin loads the model, it will automatically execute the AUTOEXEC macro. AUTOEXEC macros are useful for doing repetitive setup actions that you want every time you use a model, for example, changing printer settings if you use a special size of paper for printing reports.

They are also useful for writing "canned" application models that guide a user through a set of actions, or that allow only certain inputs and generate specific reports. The AUTOEXEC macro can present a menu so that the user never sees the Javelin menu.

If you include the @BREAKOFF command in your AUTOEXEC macro, you can prevent anyone from doing anything to a model that is not allowed

by your macro menus. Just make sure that you have thoroughly debugged your macros before doing this! If you have an AUTOEXEC macro that turns off Break, you may wonder how you can ever change that model; to change the model you have to exit from the macro, and if you haven't included this feature in your macro, no way exists to do so.

There is a way out—start with an empty model and import your model using / **F**ile **I**mport. Javelin will not run the AUTOEXEC macro when you import the model, but everything will be there for you to work with. For more information on importing, see Chapter 18, "Transferring Information Between Models." Turning off Break and having an AUTOEXEC macro should be enough to keep your users from subverting your model, unless they are actively malicious. If you need additional protection, Javelin Software's Client Services Department can tell you how to make it impossible to import your model into another model.

For completely automatic applications, you can even have Javelin load a model when it starts up. If you include the name of a model when you give the command to run Javelin, it will load the model automatically. For example, if the command JAV normally starts Javelin, you could type

```
JAV BUDGET
```

to start Javelin and automatically load the Budget model.

If that model contains an AUTOEXEC macro, Javelin runs it automatically. If you give /B on the command line after the name of the model, Javelin executes @BREAKOFF before it starts loading the model, which gives your macros complete control over what the user can and cannot do.

■ *The Worksheet Range Restrict Command*

In Javelin 1.1, when you display a worksheet and give a @PAUSE command in a macro, the user can move anywhere in the worksheet and can change the contents of the borders and any other cell. Javelin PLUS lets you protect against such changes with the commands / *Worksheet **R**ange **R**estrict and / *Worksheet **R**ange **F**ree.

Range Restrict asks you to specify a rectangular worksheet range, which you do the same way you give a range to / *Worksheet **C**opy. Thereafter, in that worksheet the user can only move the cursor to cells in the range. Range Free turns off the restriction and lets the cursor move anywhere in the worksheet. Since @PAUSE prevents the user from typing a slash, there is no

way to give a menu command and so no way for the user to escape from the restricted range. The macro can still give any command and can turn off the range restriction if needed.

You can use Range Restrict in many ways. Most commonly, you will create a data entry worksheet and restrict the user to a range of cells in which to type parameter values for the model. Be sure not to include any border cells in the range, because if you do, the user can change borders to refer to variables other than the ones you specified.

Sometimes you want to let the user look at a large worksheet and use PgUp and PgDown, but not modify anything. In that case you could create a column one character wide in the worksheet with no heading in the top border, and restrict the user to the body cells in that column. Then the cursor or PgUp and PgDown work, but there is no way to move to any other cell or change any labels or values.

A Complete Application: The Amortization Macro

■ Although Javelin provides you with the PMT function, which tells you what the payments are on a loan given an amount, an interest rate, and the term of the loan, PMT does not tell you how much of that payment is principal and how much is interest. The Javelin model to figure out those amounts is fairly simple; its formulas are shown in Figure 14.11.

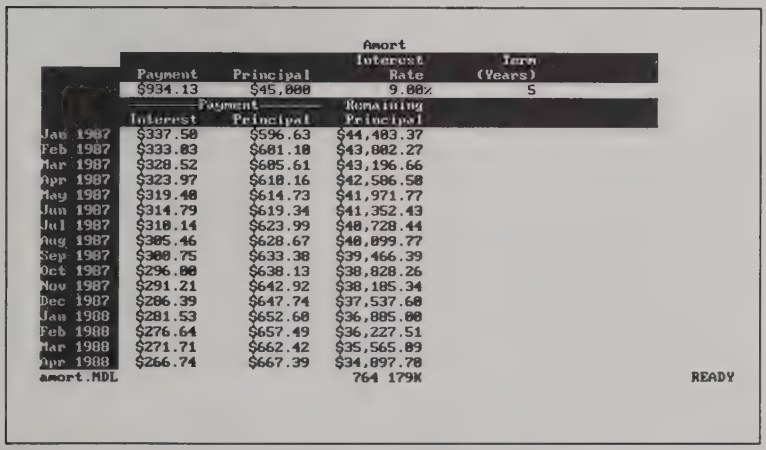
The input variables are Principal, Term, and Interest Rate. Payment is the monthly payment, using the PMT function, rounded to the nearest cent. Payment Interest, the portion of a month's payment that is interest, is the

```
Formulas for all variables:

Interest Rate = 0.09
Payment = ROUND(PMT(Principal,Interest Rate / 12,Term * 12),2)
Payment Interest = ROUND(Interest Rate / 12 * (PREVIOUS(Remaining Principal);
     Principal),2)
Payment Principal = Payment - Payment Interest
Principal = 45000
Remaining Principal = Principal - CUM(Payment Principal)
Term = 5
```

Figure 14.11: Formulas for Amortization model

monthly interest rate applied to the last month's Remaining Principal, or if there was no principal outstanding last month, to the entire loan. Remaining Principal is the entire principal minus payments to date on the principal. The Amortization worksheet, shown in Figure 14.12, summarizes the results of the model.

To change the input values for this model, all you have to do is change the values for Principal, Interest Rate, and Term and recalculate. But in order to print or graph the results, you have to know something about Javelin. The macros listed in Figure 14.13 turn this simple worksheet model into a "canned" application, ready for use by someone unfamiliar with Javelin.



	Payment	Principal	Amort Interest Rate	Term (Years)
	$934.13	$45,000	9.00%	5
	Payment		Remaining	
	Interest	Principal	Principal	
Jan 1987	$337.50	$596.63	$44,403.37	
Feb 1987	$333.03	$601.10	$43,802.27	
Mar 1987	$328.52	$605.61	$43,196.66	
Apr 1987	$323.97	$610.16	$42,586.50	
May 1987	$319.40	$614.73	$41,971.77	
Jun 1987	$314.79	$619.34	$41,352.43	
Jul 1987	$310.14	$623.99	$40,728.44	
Aug 1987	$305.46	$628.67	$40,099.77	
Sep 1987	$300.75	$633.38	$39,466.39	
Oct 1987	$296.00	$638.13	$38,828.26	
Nov 1987	$291.21	$642.92	$38,185.34	
Dec 1987	$286.39	$647.74	$37,537.60	
Jan 1988	$281.53	$652.60	$36,885.00	
Feb 1988	$276.64	$657.49	$36,227.51	
Mar 1988	$271.71	$662.42	$35,565.09	
Apr 1988	$266.74	$667.39	$34,897.70	

amort.MDL 764 179K READY

Figure 14.12: Summary worksheet for Amortization model

```
Macro AUTOEXEC:

@REM This macro makes this model self-running.
@F7 Main @ENTER
@SLASH Q Y Y      @REM Quit when done.


Macro Main:

@SILENT Initializing ...
@SLASH W 1      @REM Do all this in one window.
@SLASH V W Inputs @ENTER
@ENDSILENT
```

Figure 14.13: Macros in Amortization model

```
@LABEL loop
   @REM Ask the user what he wants to do next.
   @SELECT Amortization system
      @CHOICE change Change amount, term, or rate
      @CHOICE table Monthly amortization table
      @CHOICE graph  Graph of amortization
      @CHOICE quit   Exit this program
   @ENDSELECT

@LABEL quit
   @EXIT         @REM All done.

@LABEL change
   @F7 Accept @ENTER
   @JUMP loop

@LABEL table
   @SELECT Do you wish to print or view the table?
      @CHOICE tableview View the table on the screen
      @CHOICE tableprint Print the table on the printer
   @ENDSELECT
   @JUMP loop   @REM User changed his mind.

@LABEL tableview
   @F7 Display @ENTER        @REM Display macro displays the monthly table.
   @JUMP loop

@LABEL tableprint
   @SLASH V W Amort @ENTER    @REM Bring up the worksheet.
   @SLASH P C Y               @REM Print it.
   @JUMP loop

@LABEL Graph
   @SELECT View or Print the graph?
      @CHOICE graphview View the graph on the screen
      @CHOICE graphprint Print the graph on the printer
   @ENDSELECT
   @JUMP loop   @REM User changed his mind.

@LABEL graphview
   @SLASH V G Amort @ENTER   @REM Bring up the graph.
   @READ                     @REM Give the user a chance to look at it.
   @JUMP loop

@LABEL graphprint
   @SLASH P G Pr-amort @ENTER
   @JUMP loop


Macro Accept:

@REM Input the parameters for the mortgage.

   @SLASH V W Inputs @ENTER  @REM Bring up the inputs worksheet.
   @RIGHT @DOWN              @REM Move to first data cell.
   @PROMPT Enter principal, term, date.  Press F7 when done.
   @PAUSE
   @REM Have to recalculate after a change.
   @SLASH V W Calculation @ENTER     @REM Put up apology to stall user...
   @F9                               @REM ... while calculating.


Macro Display:

@REM Display the amortization table.
@REM Recalculation can take a while, so warn the user.
   @SLASH V W Amort @ENTER  @REM Bring up the worksheet.
   @PROMPT Press PageUp and PageDown to see more, press F7 to continue.
   @PAUSE @ESC          @REM The ESC throws away any junk the user might type.
```

Figure 14.13: Macros in Amortization model (continued)

These macros fit together as follows:

- AUTOEXEC runs as soon as the model is loaded.
- AUTOEXEC runs the Main macro.
- When the Main macro is finished, it exits from Javelin without saving the model.

The Main macro is a good example of how to set up a "driver" macro that controls a macro system in Javelin. It has three essential parts: an initialization section, which silently sets up the model so it is ready for the user's work; a menu section, which figures out what the user wants to do; and a section for each menu selection. In this macro, if the user hits Esc, the Main macro continues after @ENDSELECT, exits back to the AUTOEXEC macro, and stops running. Figure 14.14 shows the menu as it appears on the Javelin screen.

The Change selection is an example of one macro calling another. The steps in the Accept macro are not very complicated, and there is no technical reason why they couldn't be included in the Main macro, except that this would make the Main macro longer and harder to understand. In general, it's a good idea to break up your application logically into several macros. The Accept macro shows the user a worksheet on which only the input values are shown and @PAUSEs to let them change these values. It then recalculates the model, putting a message on the screen to tell the user what it is doing. The screen it displays is shown in Figure 14.15. When Javelin reaches the end of the Accept

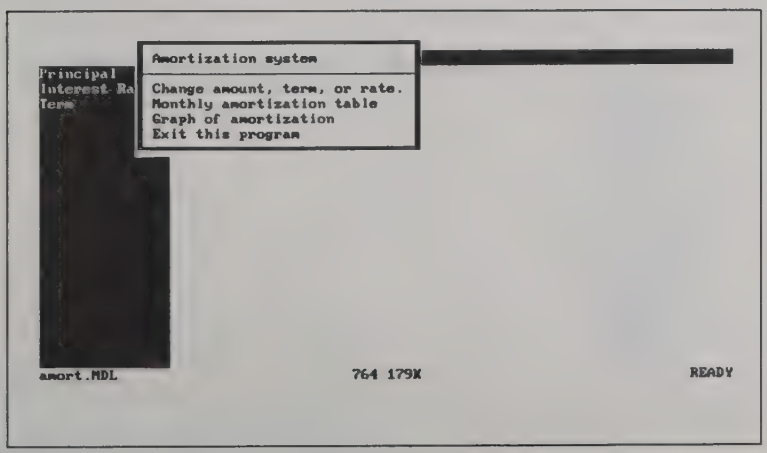


Figure 14.14: Main menu in Amortization model

macro, it returns to the Main macro that called Accept and jumps back to the loop, which asks the user what to do again.

To make this macro more bulletproof in Javelin Plus, you can use / *Worksheet **R**ange **R**estrict in the Inputs worksheet to limit the user to the three cells that contain the values for the principal, the interest rate, and the length of the loan. You could also create a column one character wide in the Amortization worksheet and restrict the user to the cells in that column to prevent changing the model.

The Table and Graph sections of the macro are examples of additional macro programming that do not really require a separate macro. They are short and very simple, each one asking merely whether to output to the screen or to the printer. You could put them in separate macros if you wish to; it is an aesthetic decision.

Good Macro Practices

■ As with a model, the difference between a "good" and a "messy" macro is as much aesthetic as technical. This section does not apply to convenience macros, which are used briefly and discarded. But if you create personal macros you expect to use often, and particularly if you create applications, use these pointers.

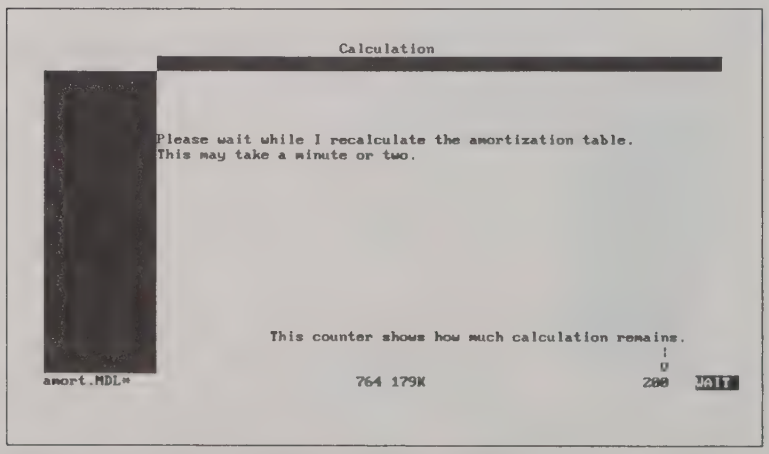


Figure 14.15: Calculation screen in Amortization model

■ Make Your Macros Robust

Don't assume that Javelin is in a particular view or even that it has two windows on the screen when your macro starts. If the windows are important to your macro, set them up explicitly by doing the command sequence / **W**indow **1** / **W**indow **2**. If the user has adjusted the size of the windows, this returns them to the Javelin default. It also puts the cursor in the lower window and is the only way for a macro to ensure that it is in a particular window.

Macros should also use the @ERROR feature to check that they are in fact working. In a closed application, if a macro encounters an error it can't understand, you should probably have it save itself and print a message asking the user to send you the aborted model so you can fix it. If you value results and your macros are complex, make sure to check whether an error has occurred.

■ Make Your Macros Readable

Macros can be hard to read, but it is easier to see what is happening by inserting spaces and blank lines to break up the text.

Another good idea when writing macros is to include remarks that tell you what each part of the macro is for. In Javelin 1.1 and later, the @REM macro command lets you include such remarks. When Javelin encounters the @REM command in a macro, it ignores the rest of the text on that line, so you can put whatever notes will be useful to you. To include a really long comment, put a @JUMP before it and a @LABEL after it to tell Javelin to jump around it.

Here are some other good methods for making your macros readable:

- Put each Javelin command on a new line.
- When recording a macro that gives commands from Javelin menus, type the first letter of each command, rather than moving the cursor to the menu choice and pressing ⏎. This makes the macro easier to read in the Macro view and makes it run much more reliably.
- Indent @IF/@THEN/@ELSE/@ENDIF blocks to show which @ELSE and @ENDIF go with each @IF.
- Indent the menu @CHOICES under the @SELECT command so you can see what's part of the menu.

- Indent loops so that the body of the loop is indented uniformly.
- Start @LABELs at the left margin to make them easier to find.
- Use several small macros rather than one large one. This is the most important rule of all—a ten-page macro, no matter how carefully indented and annotated, is usually baffling.

■ *Protect Your Model*

With the @BREAKOFF command and the AUTOEXEC macro, your model can become a closed application to the user. The following pointers will help make your application more secure.

You will frequently use @PAUSE in your macros to let the user wander around a view (usually Table or Worksheet) to look at a model's results. This means that the user could change existing formulas in the model. You probably don't want anyone to be able to do this, and you can prevent it by Locking (on the settings sheets) all the variables you don't want the user to change. Because the user cannot give commands to view the Variable settings sheet, there is no way of unlocking the variable and your formulas are safe from harm. In Javelin PLUS, you can also use / *Worksheet **R**ange **I**nput to restrict where a user can type.

Don't use the @READ command if you think your user might be malicious. For instance, if you want to let your user view one of several worksheets, you might be tempted to use

```
/ V W @SPELL @READ
```

The user could then move the cursor onto a desired worksheet name and hit ⟵. Your macro could hit ⟵ to select that worksheet. However, a malicious user could hit Esc and select some other command, thus breaking your application. If you expect hostile users and have to let them make these kinds of selections, write a menu yourself that allows them select among different worksheets, graphs, and variables.

■ *Notes on the @PAUSE Command*

If you expect the user to move around (using cursor keys) while the macro pauses, it is a good idea to have some notation on the screen reminding how to continue with the macro. Javelin's message *Press F7 to continue*

disappears off the top of the screen as soon as the user hits any key. A favorite technique of ours works this way:

- Adjust the screen so that one window is only two lines high (the smallest Javelin allows).
- Define a variable called *using this model.*
- In the Notes view, type a one-line message like *Press the F7 key when you are finished viewing the table.*
- Put the cursor in the other window.

This way, the user always has some idea of what to do next. You can even have several such messages in the note and show a different one depending on what the model is doing. Figure 14.16 shows an example of this technique used in the Amortization model.

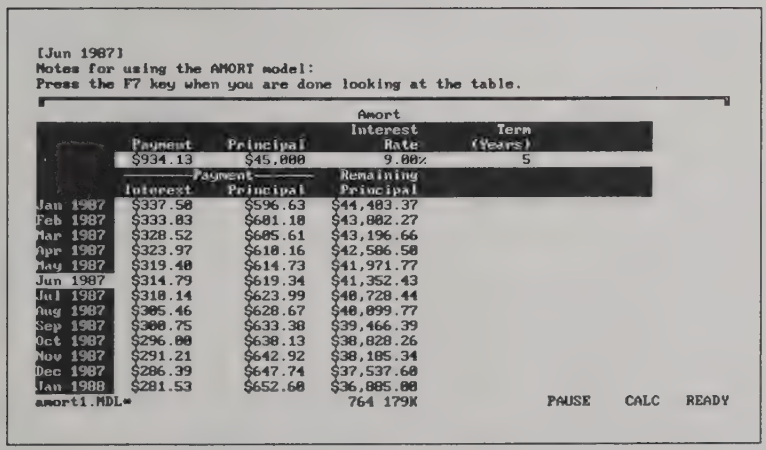


Figure 14.16: Message window using the Notes view

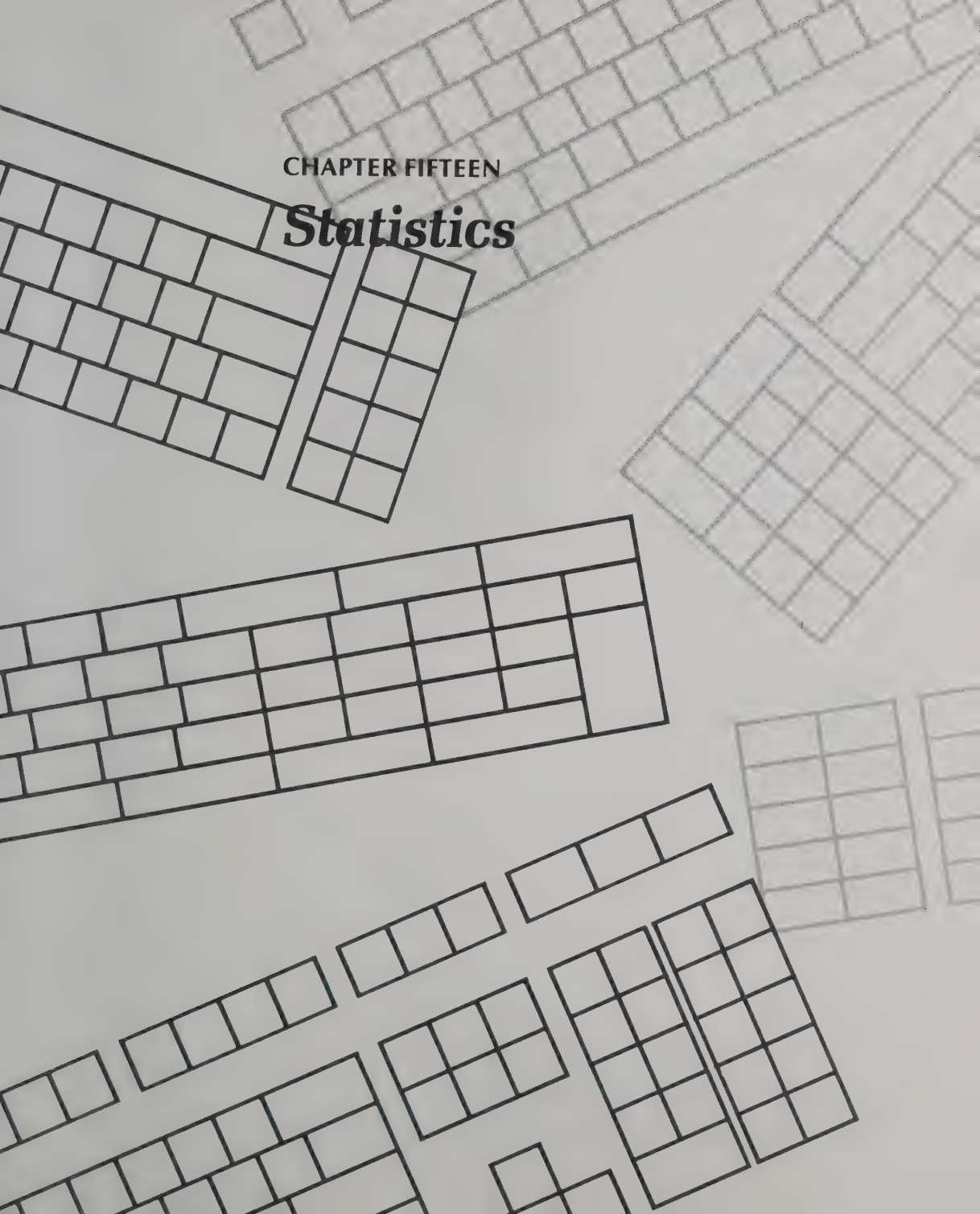
Conclusion

■ Javelin's macro facilities are useful for many kinds of projects, from building models more easily to creating full-blown applications. Simple macros are easy to define, since Javelin can record keystrokes as you work,

and easy to run. Use macros often when creating large models—you will find that they save you considerable time.

Javelin's menu and pause features allow the applications you create to take advantage of Javelin's user interface without forcing your users to learn about the program itself. Javelin's macro language is rich enough that you can do almost anything, although since macros can't do calculations, complex applications may have to be a little devious to achieve their results. The Histogram model in this chapter illustrates this limitation and how to work around it.

You may not need all the considerable power that Javelin macros have, but familiarity with what they can do will make you a much more effective Javelin user.



CHAPTER FIFTEEN

Statistics

Javelin's handling of statistics represents a major temptation. This good set of basic statistical functions waits to lure the unwary into a statistician's nightmare: using the wrong test on the wrong data. In this chapter, we will describe how Javelin's statistical functions work and how they can be combined. We will not tell you when it is appropriate to use them; that's the statistician's job. It is always up to you to check that your model's results correctly represent the situation you are modeling, and that is particularly true of statistical analyses.

Javelin can do two types of statistical analysis: *least-squares linear regression* and *analysis of variance*. In Javelin PLUS, trigonometric functions were added for engineers. They're not really statistical functions, but we'll discuss them here as well.

We'll use the ECON model from Chapters 8 and 12 to illustrate the use of least-squares linear regression, predicting growth in imports and exports based on a correlation with GNP growth. As promised at the end of Chapter 12, we'll graph the predicted values and the actual values. Later, we'll look at some sales data using analysis of variance to compare the performances of three different salesmen.

Least-Squares Linear Regression Analysis

■ We introduced linear regression at the end of Chapter 12, when we looked at the relationship between Imports, Exports, and Growth in the U.S. economy. Figure 15.1 graphs U.S. exports and imports as a function of U.S. GNP; that is, for a given GNP growth, it shows the associated export growth and import growth for that year. There *seems* to be some relationship—higher GNP growths have higher import and export growths, and lower (or negative) GNP growths have lower (or negative) import and export growths. Furthermore, it *seems* (from looking at the graph) that imports "track" GNP better than exports because they are less scattered and group more tightly around a straight line running from the lower left of the graph to the upper right.

That hypothetical line is the subject of linear (hence the name) regression analysis. It allows you to answer questions like

- What is the formula for this line?

- How close are the import numbers to that line, measured numerically?
- How close are the export numbers to that line?
- Based on that line, if next year's GNP grows at a certain percent, what is likely to happen to imports and exports?

■ *Predicting the Value of One Series Based on Another*

The formulas for the ECON model are listed in Figure 15.2. We are concerned with three sets of formulas for predicting import growth (based on GNP growth), export growth (based on GNP growth), and growth in a hypothetical import tax (based on import growth). We'll use graphs to illustrate the use of the statistical functions, starting with a graph of imports versus GNP (see Figure 15.3).

First let's see how well the actual import growth matches the predicted import growth based on GNP growth. The Javelin PREDICT function calculates the prediction for us:

- PREDICT (x series, y series, x value): Predicted value of the y series for the period when x equals the x value specified, using the least-squares line. The x value may be a constant or a time series.

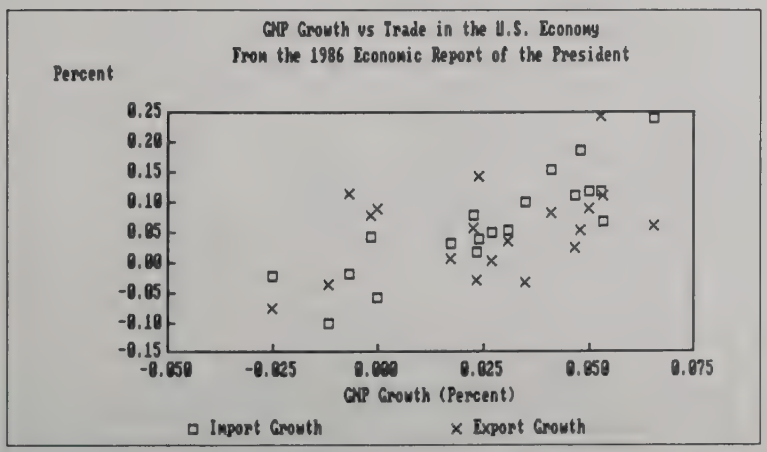


Figure 15.1: Import and Export Growth plotted against GNP Growth

PREDICT returns a constant if the value for which you want a prediction is a constant, otherwise it returns a time series.

Let's use PREDICT and then explain it in more detail. To calculate Predicted Import Growth based on GNP Growth, the formula is:

Predicted Import Growth = PREDICT(Pct Change GNP Adjusted, Pct Change Imports, Pct Change GNP Adjusted)

Figure 15.4 shows the line that this prediction draws, based on the GNP growths you actually have. Your prediction and your actual data run for the same times because you told PREDICT to base its predictions on Pct Change GNP Adjusted. If you had told it to use some series of hypothetical future GNP growths, you would have gotten Import data corresponding to those growths. But before getting carried away predicting the future, you should know how close Javelin's prediction is when you actually know the data. For that matter, how did Javelin draw that line?

Javelin computes linear regression by using the least-squares method. That is, it calculates the line so that the sum of the squares of the distances between the observed points and the line is minimized. This means that if a graph of the two series were actually a straight line, Javelin's regression line would fit perfectly. Hypothetical import tax formulas are an example. The formula defining Import Taxes is

Import Taxes = -.1 * Imports
Pct Change Import Tax = PCT(Imports) * -1

```
Formulas for all variables:

Adjusted GNP = GNP/GNP Deflator * 100
Adjusted GNP in Billions = INT(Adjusted GNP)
Adjusted GNP Rounded = ROUND(Adjusted GNP,1)
Change GNP Adjusted = CHANGE(Adjusted GNP)
Change Unemployment = CHANGE(Unemployment)
Federal Defense Spending = < Yearly data: 1966 - 1985 >
Federal NonDefense Spending = < Yearly data: 1966 - 1985 >
Federal Spending = SUM(Federal Defense Spending,Federal NonDefense
     Spending)
GNP = < Yearly data: 1966 - 1985 >
GNP Deflator = < Yearly data: 1966 - 1985 >
GNP Growth = MAX(Change GNP Adjusted,0)
Imports = < Yearly data: 1966 - 1985 >
Pct Change GNP Adjusted = PCT(Adjusted GNP)
Predicted Growth = SELECT(Recession, 0.05, 0.02, 0)
Problem = IF(OR(Change GNP Adjusted < 0,Change Unemployment > 0),1,0)
Recession = IF(Change GNP Adjusted < 0,IF(Change Unemployment > 2,3,2),1)
Unemployment = < Yearly data: 1966 - 1985 >
```

Figure 15.2: Formulas for ECON model

Pct Change Import Tax has a linear relationship to Pct Change Imports: when Pct Change Imports doubles, Pct Change Import Tax doubles; when Pct Change Imports falls by one-third, Pct Change Import Tax falls by one-third. To see this relationship, let's graph Pct Change Imports versus Pct Change Import Tax (see Figure 15.5).

The relationship is linear. Javelin will find it very easy to predict Import Tax based on Imports. The formula is

Predict Import Tax = PREDICT(Pct Change Imports, Pct Change Import Tax, Pct Change Imports)

How well do the predicted taxes match the actual taxes? In Figure 15.6, the predicted and observed points are superimposed. When the relationship between the two variables is in fact linear, least-squares linear regression gives you a perfect prediction.

Let's take a third example, predicting exports based on GNP growth. The formula is the same as for Import and Import Tax Growths:

Predicted Export Growth = PREDICT(Pct Change GNP Adjusted, Pct Change Exports, Pct Change GNP Adjusted)

The Javelin PLUS XY graph shows both the actual Pct Change Exports and the least-squares line Predicted Export Growth plotted against Pct Change GNP Adjusted.

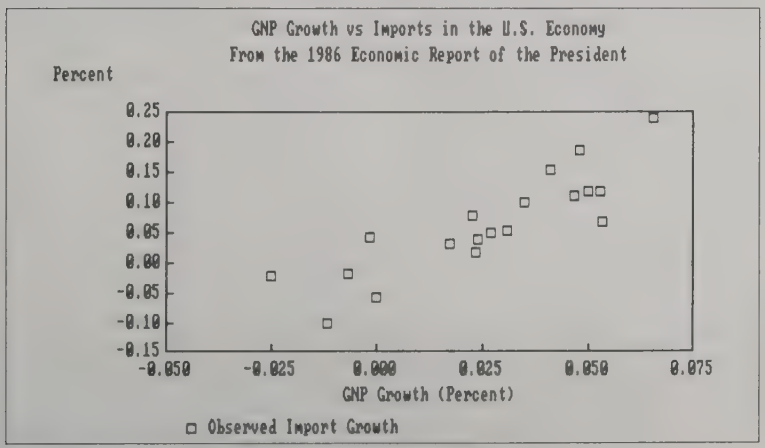


Figure 15.3: Import Growth plotted against GNP Growth

Compare Figure 15.7 with Figures 15.6 and 15.4. Figure 15.6 shows variables with a perfect linear relationship, and there is a perfect correlation between prediction and reality. In Figure 15.4, there is clearly a close relationship. In Figure 15.7, although there is a relationship, it is not clear how strong it is.

A common statistical measure used to tell how well two series of data relate is *Pearson's r correlation coefficient*. The CORR function in Javelin computes Pearson's r for two series:

- CORR (x series, y series): Calculates the correlation coefficient for two series, a constant between 1 and −1 that tells how closely the series are related linearly. If the correlation is close to 1, the relation is strong. If the correlation is close to −1, there is an inverse relation. If the coefficient is close to 0, there is little relation.

You can calculate the correlation for the predictions you have done, as in Figure 15.2:

```
Correlation Exports GNP = CORR(Pct Change GNP Adjusted, Pct
                          Change Exports)
Correlation Imports GNP = CORR(Pct Change GNP Adjusted, Pct
                          Change Imports)
Correlation Import Tax Imports GNP = CORR(Pct Change Imports,
                                     Pct Change Import Tax)
```

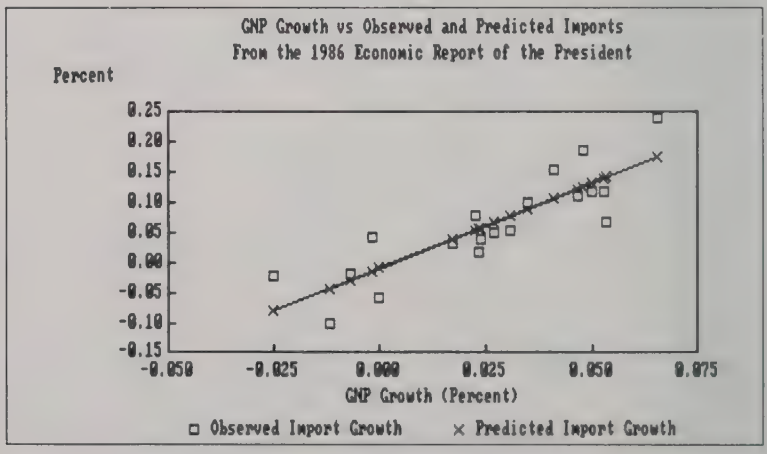


Figure 15.4: Linear regression of Import Growth vs. GNP Growth

The values for these variables are shown in Figure 15.8. As expected, correlation between Import Growth and Import Tax Growth is 1, a perfect correlation. When imports go up, import taxes go up. The correlation coefficients for Import Growth vs. GNP Growth and for Export Growth vs. GNP Growth are both positive, but imports correlate with GNP much more strongly (.865) than do exports (.405).

How significant is this difference? It depends entirely on your data and the field you are in. In the physical sciences, correlations below .9 typically are not considered very significant. In economics, correlation coefficients above .8 are considered significant. In the social sciences, correlations as low as .4 can be useful.

Note that the correlation coefficient does not imply causality—it does not say that imports went up *because* GNP went up; it tells you that *when* GNP went up, imports went up. The causality could run in either direction, or both could be caused by something totally different.

■ *Calculating the Correlation and the Equation of the Line*

A line is defined by the formula

$$Y = (M * X) + B$$

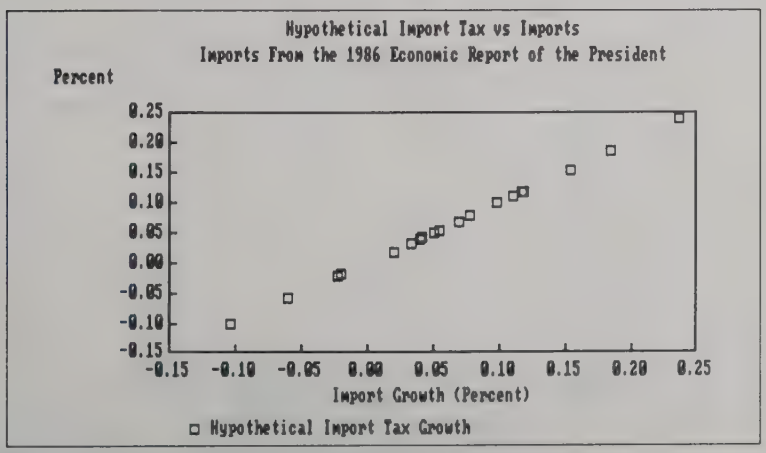


Figure 15.5: Hypothetical Import Tax Growth vs. Import Growth

where M is the *slope* and B is the *y-intercept* of the line, that is, the y value when x is zero. The Javelin functions SLOPE and INTERCEPT give you the respective values for the regression line:

- INTERCEPT (x series, y series): Calculates the y-intercept of the least-squares line for two series.
- SLOPE (x series, y series): Calculates the slope of the least-squares line for the two series.

Both functions return a constant. These formulas find the slope and intercept of the line for GNP Growth vs. Import Growth:

```
M = SLOPE(Pct Change GNP Adjusted, Pct Change Imports)
B = INTERCEPT(Pct Change GNP Adjusted, Pct Change Imports)
```

For each one-percent change in GNP, imports will change by M percent; when GNP does not grow at all, imports will change by B percent.

■ *Predicting the Value of One Series Against Time*

So far, you have related two series by asking, If I know something about GNP growth, what can I infer about imports or exports? A different question is, What is the general trend of the GNP? You can also use least-squares

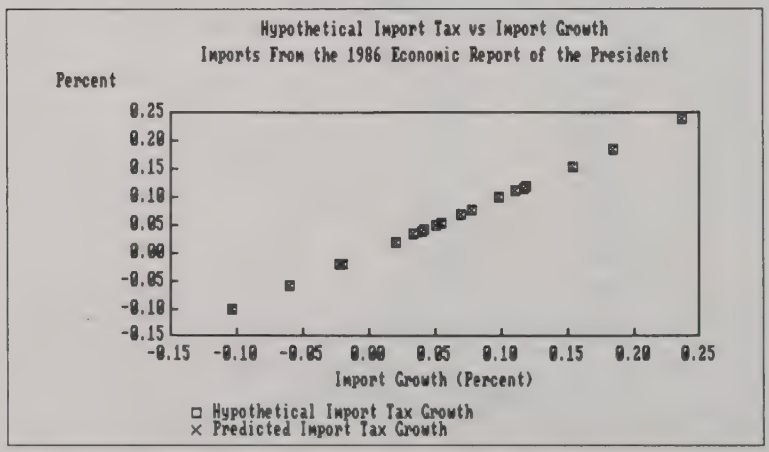


Figure 15.6: Predictions about hypothetical Import Tax Growth

linear regression to look at the behavior of a single time series over time.

Javelin provides trend functions that are specialized versions of the regression functions. In the trend functions, you specify only the x series. The y series is time. The TREND function, like PREDICT, computes the least-squares trend line of its input versus time.

- TREND (series, begin date, end date): Calculates the predicted values based on the least-squares line for the series. The result is a series of predicted values for the range of dates you specify. Begin date and end date are optional. If you leave them out, Javelin uses the beginning and ending dates of the first argument.

This formula shows how GNP acts over time:

Predicted Adjusted GNP = TREND(Adjusted GNP)

Figure 15.9 shows Adjusted GNP and its trend line. GNP tracks the trend line very closely. Using the TRENDCORR function (analogous to CORR above), you can find out how closely it tracks:

- TRENDCORR (series): Calculates the correlation coefficient for the series compared to its trend line over time, a constant between 1 and −1. It tells you how closely the series is related to time. Like CORR, values near 1 or −1 indicate a strong relation, and values near 0 indicate little relation.

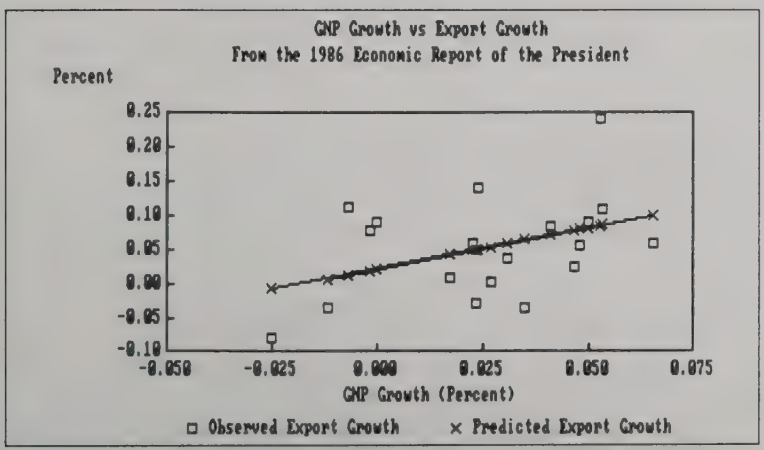


Figure 15.7: Linear regression of Export Growth vs. GNP Growth

The correlation between the trend line and the actual data is .987, a very close correlation. This means that, if the future behaves like the past, and if you have selected a representative sample of years, predictions made about GNP using the trend line of past GNP will be quite accurate.

TREND takes optional date arguments to specify the time range for making predictions. The variable Future GNP predicts GNP between 1986 and 1996.

Future GNP = TREND(Adjusted GNP, 1986, 1996)

Be careful when using trend lines to make predictions. Normally cautious people have made astoundingly naive predictions based on trends. Two of our favorites are an 1880 prediction that by 1930 New York City would be unlivable because it would be 20 feet deep in horse manure (straight-line prediction based on number of horses) and a 1920 prediction that by 1970 one out of every two Americans would work for the telephone company (straight-line prediction based on long-distance telephone usage and employment). Straight-line predictions may be accurate for the short and perhaps the medium term, but are hardly ever accurate for the long term.

You can also find the slope and intercept of the trend line:

- TRENDSLOPE (series): The slope of the least-squares line for the series trend line over time, which indicates the predicted increase of the series in each period.
- TRENDSTART (series): The y-intercept of the least-squares trend line for the series.

■ *More Sophisticated Analysis*

You can combine Javelin's built-in statistical functions to do more complicated analyses. One such useful statistic is serial autocorrelation of error.

```
Correlation
  Imports GNP                0.865
  Exports GNP                0.405
  Import Tax Imports         1.000
```

Figure 15.8: Worksheet showing correlation coefficients

When you draw a least-squares line between two variables or between a variable and time, you assume that the underlying relationship is linear and that any deviation from linearity is due to random variation or measurement error. This means there should be no correlation between the deviation at one date and that at the next. If there is, it means that rather than deviating at random, the error tends first to one side of the trend line and then to the other, which suggests that the true relationship may not be linear.

Previously you used TRENDCORR to find that the correlation between GNP and time is .987, which is very high. Now let's look at the autocorrelation of error, by calculating the correlation of the error against the error in the previous period:

```
GNP Predicted = TREND(Adjusted GNP)
GNP Pred Error = Adjusted GNP - GNP Predicted
GNP Autocorr = CORR(GNP Pred Error, PREVIOUS(GNP Pred
               Error))
```

GNP Pred Error tells how far the predicted value for each date is from the actual value. GNP Autocorr finds how great the correlation is from one error value to the next. In this case it is .4, not a very high value but enough to suggest that despite the high trend correlation you got, there may be important nonlinear components.

A statistical analysis that you cannot create from the built-in functions is *multiple regression,* a technique that simultaneously regresses several y

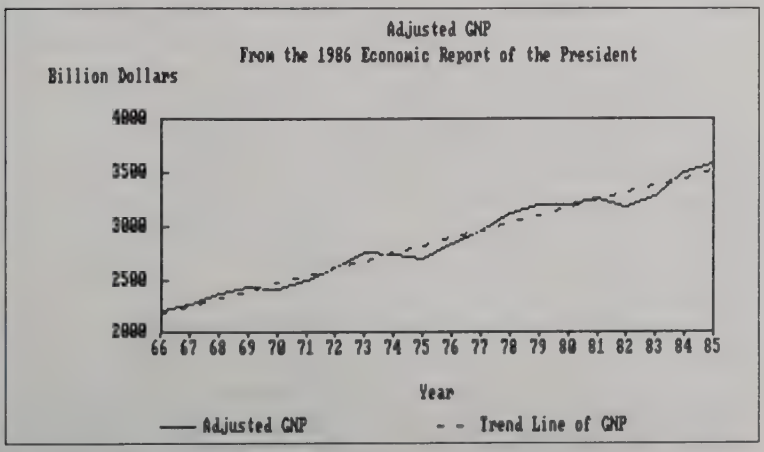


Figure 15.9: Trend line of Adjusted GNP

series against a single x series. Fortunately, in Javelin PLUS multiple regression is available in a new snap-in building block.

Analysis of Variance

■ Analysis of variance is a way of analyzing a set of values to determine the average value and how closely the rest of the values are clustered around the average. You have already used the TAVG function for calculating the average value for a series. Here are Javelin's functions for finding how the values cluster:

- STDEV (series): Calculates the standard deviation of the series.
- VAR (series): Calculates the variance of the series.

Variance and standard deviation measure the same thing. In fact, the standard deviation is just the square root of the variance. For most applications, the standard deviation is more useful because it is expressed in the same units as the input. The important thing to know is that for any series, about 63 percent of all data points lie within one standard deviation of the average of the series.

Table 15.1 shows three years of sales data for the three salesmen in an office. We've adjusted for seasonal variations by expressing all the sales numbers relative to the overall average sales for the company. A value of 100 means that the salesman sold the average amount.

A quick look suggests that Jones is the best salesman, since he had months as good as 147, while Smedley is the worst, as his best month was only 119. But let's take a closer look. Compute the maximum, minimum, and average monthly sales as well as the standard deviation of sales for each salesman using Javelin's TMAX, TMIN, TAVG, and STDEV functions. The formulas are listed in Figure 15.10, and Figure 15.11 shows a worksheet with the results.

When you look at the analysis, a different picture emerges. Although Jones has the best months, he also has the worst, selling as little as 22 in one month. A look at the averages and standard deviations shows that Jones has the lowest average sales and also the highest deviation, while Smedley has the highest average sales and the lowest deviation. Smedley reliably

	SALES		
	Smith	**Jones**	**Smedley**
Jan 1984	90	52	106
Feb 1984	92	147	106
Mar 1984	75	22	102
Apr 1984	109	67	106
May 1984	102	83	102
Jun 1984	90	107	103
Jul 1984	84	106	94
Aug 1984	74	66	101
Sep 1984	94	104	108
Oct 1984	103	113	112
Nov 1984	84	94	104
Dec 1984	119	105	112
Jan 1985	77	83	99
Feb 1985	108	91	119
Mar 1985	80	121	107
Apr 1985	132	145	102
May 1985	97	133	112
Jun 1985	109	129	113
Jul 1985	105	91	108
Aug 1985	96	129	101
Sep 1985	97	48	104
Oct 1985	96	63	103
Nov 1985	95	86	105
Dec 1985	106	83	104
Jan 1986	105	76	109
Feb 1986	104	47	105
Mar 1986	99	94	92
Apr 1986	110	85	116

Table 15.1: Three years of sales data

May 1986	98	102	116
Jun 1986	82	70	108
Jul 1986	104	71	102
Aug 1986	96	91	99
Sep 1986	105	96	104
Oct 1986	115	114	107
Nov 1986	111	105	104
Dec 1986	97	106	111

Table 15.1: Three years of sales data (continued)

turns out above-average results, while Jones is erratic and his good months do not make up for his bad ones. Smedley is the real star.

Random Numbers

■ You may want Javelin to make up data to test the behavior of a model when you do not know the actual arguments, or to do *Monte Carlo* simulations where you want to see the result of your model given a variety of possible inputs. You can use Javelin's RANDOM function:

- RANDOM ( ): Produces a number between zero and one that is uniformly distributed; that is, all values between zero and one are equally likely. RANDOM takes no arguments and returns a constant.

To get a monthly series of random numbers, you can use the function MONTHLY(RANDOM), or a similar expression using BEGIN, END, or RANGE. Every time Javelin evaluates the RANDOM function it comes up with a new value, so that MONTHLY(RANDOM) comes up with a new random number for each month.

For random numbers in a range other than zero to one, you can multiply the result of RANDOM by the desired range and add it to the desired minimum value. For example, to get the effect of rolling a pair of dice, start with

```
Die A = INT(1 + 6 * RANDOM( ))
Die B = INT(1 + 6 * RANDOM( ))
Dice Roll = Die A + Die B
Snake Eyes = (Dice Roll = 2)
```

To get a series of 100 dice rolls, using the custom time period A, replace the RANDOM function with

```
RANGE(1 A, 100 A, RANDOM( ))
```

There is another way to get random numbers:

- NORMAL ( ): Produces a normally distributed value with a mean of zero and a standard deviation of one. NORMAL takes no arguments and returns a constant.

```
Formulas for all variables:

Average Jones = TAVG(Sales Jones)
Average Smedley = TAVG(Sales Smedley)
Average Smith = TAVG(Sales Smith)
Max Jones = TMAX(Sales Jones)
Max Smedley = TMAX(Sales Smedley)
Max Smith = TMAX(Sales Smith)
Min Jones = TMIN(Sales Jones)
Min Smedley = TMIN(Sales Smedley)
Min Smith = TMIN(Sales Smith)
Sales Jones = < Monthly data: January 1984 - December 1986 >
Sales Smedley = < Monthly data: January 1984 - December 1986 >
Sales Smith = < Monthly data: January 1984 - December 1986 >
Std Dev Jones = STDEV(Sales Jones)
Std Dev Smedley = STDEV(Sales Smedley)
Std Dev Smith = STDEV(Sales Smith)
```

Figure 15.10: Formulas for sales analysis

	Smith	Jones	Smedley
Max	132	147	119
Min	74	22	92
Average	98.3	92.4	105.7
Std Dev	12.5	27.6	5.7

Figure 15.11: Results of sales analysis

The normal distribution is the familiar bell-shaped curve. You get a series of normal values the same way you get a series of random values, using MONTHLY, RANGE, BEGIN, or END. For example, to get a monthly series of random values with mean 10 and standard deviation 6, enter

```
Samples = 10 + 6 * MONTHLY(NORMAL( ))
```

The normal distribution is useful for producing realistic time series. To simulate a stock price that increases on the average 3 percent per month, has a month-to-month standard deviation of 20 percent, and starts out at a price of 100, use the formula

```
Price = PREVIOUS(Price * 1.03) * (1 + 0.2 * NORMAL( )); 100 @
        Jan 87;
```

In this formula, PREVIOUS(Price * 1.03) computes a 3-percent increase over the previous month. The expression (1 + 0.2 * NORMAL( )) produces a series with mean 1 and standard deviation .2 or 20 percent. Multiplying the two expressions together randomly adjusts the smoothly increasing price series to give the desired result.

Trigonometric Functions

■ Javelin PLUS includes a full set of trigonometric functions. They are useful mostly for doing scientific rather than financial analysis, although occasionally they can help analyze seasonal or periodic behavior.

- SIN (x): Sine of x.
- COS (x): Cosine of x.
- TAN (x): Tangent of x.
- ASIN (x): Arc-sine of x.
- ACOS (x): Arc-cosine of x.
- ATAN (x): Arc-tangent of x.
- ATAN2 (x,y): Arc-tangent of x/y.

The inputs to SIN, COS, and TAN and the outputs of ASIN, ACOS, ATAN, and ATAN2 are expressed in radians rather than in degrees. You can make impressive graphs by plotting sines and cosines against each other, such as the graph in Figure 15.12.

Conclusion

■ Javelin provides a variety of statistical functions that you can use in your models. The linear regression functions let you use data from one series to predict unknown values in another series, and to see to what degree the first series seems to affect the second. The correlation functions let you test two series against each other to see if the two are linearly related, and how close the predictions from linear regression of the two series are.

The trend functions are specialized versions of the linear regression and correlation functions. They let you extrapolate a series' growth over time and test how linear the growth over time is. The variance and standard deviation let you test how closely a series of data clusters around its average.

The random number functions let you invent data, for creating test data for a model or data for possible scenarios for a model with uncertain factors. Finally, the trigonometric functions are useful for engineering calculations and graphics.

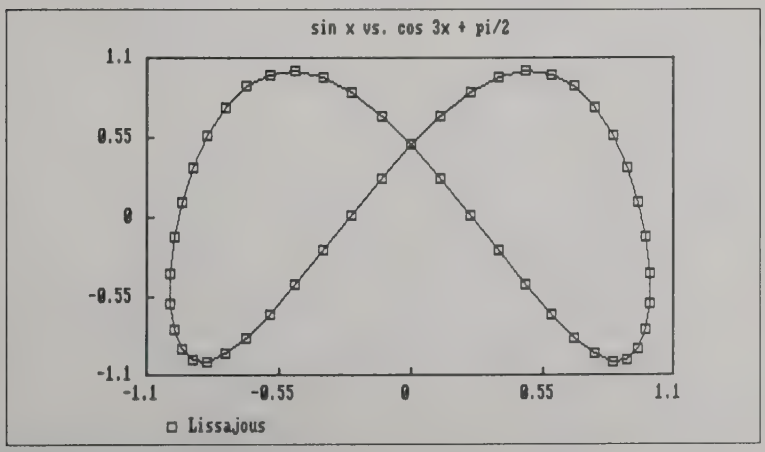
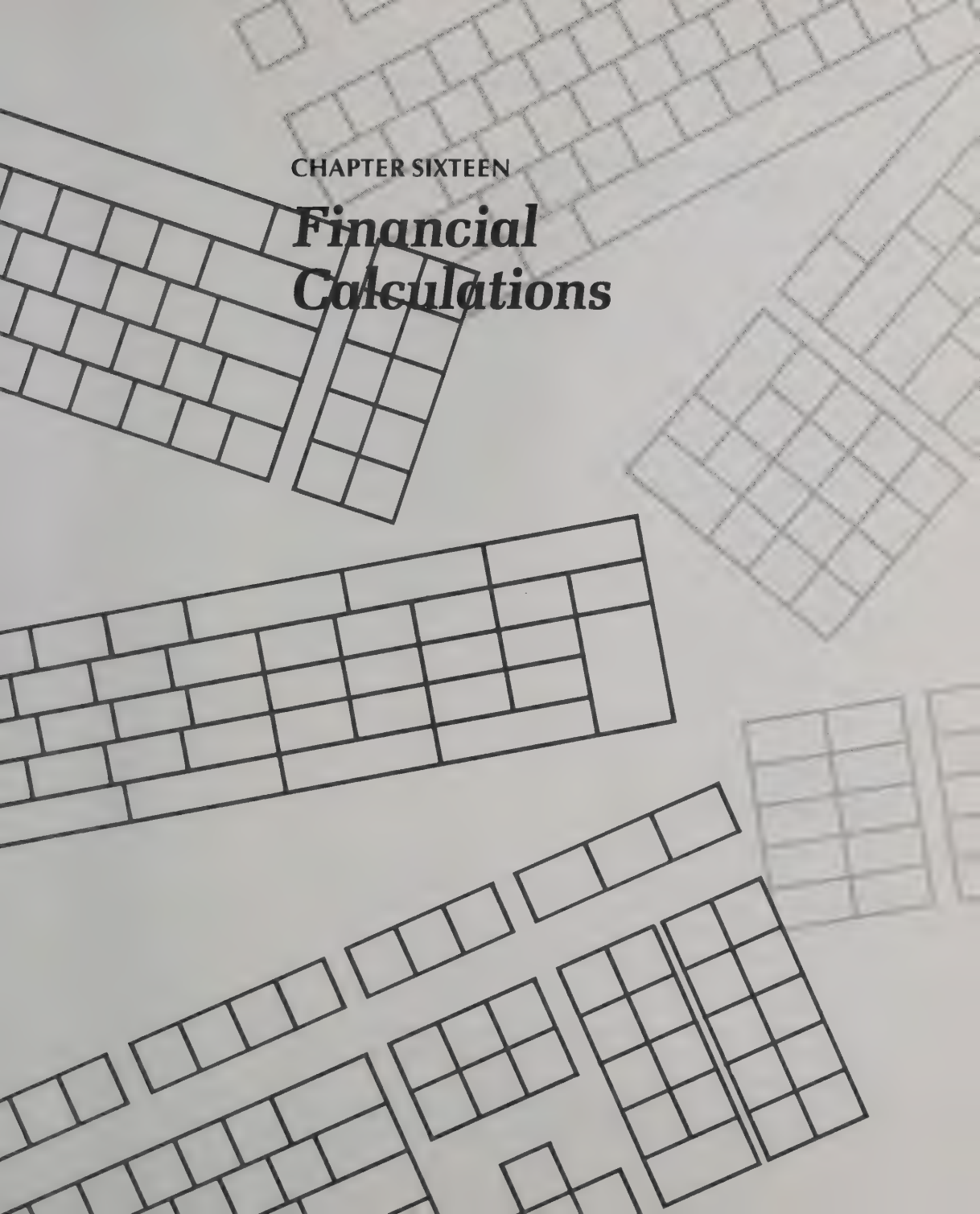


Figure 15.12: Sine plotted against cosines



CHAPTER SIXTEEN

Financial Calculations

Javelin includes a number of functions designed for financial applications. These functions fall into three groups:

- Present and future values, for modeling the values of income flows and determining effective interest rates.
- Depreciation, for calculating the depreciation values for an asset for a series of periods, using one of a number of standard methods.
- Bond pricing, for calculating bond prices or yields based on face value, coupon rate, and years to maturity.

Of these three classes of functions, the present and future value functions are by far the most generally useful; we will go over them in some detail. Depreciation and bond pricing can be extremely convenient, but if you need them, you also know how they work. Therefore we will review them briefly here.

In this chapter, you will set up a model to analyze an investment in a money fund, to find out how the fund is performing. It tracks multiple share purchases and redemptions and takes into account that the share price of the fund is changing. As in Chapter 15, there is much more to finance than we can discuss in the context of Javelin. But we will give you a sense of when to use these functions.

Present and Future Values

■ You commonly think of an interest rate as the percentage you expect the money you have saved to grow each year, or the percentage you have to pay to borrow money. To use present and future values, you need a different way of thinking about interest rates. If you receive $100 today, you decide whether to spend or save it. If you decide to save it (at, say, 15 percent per year), in effect you say that $115 in one year's time is worth more to you than $100 today. But if your investment would yield only $110 in a year's time, a rate of 10 percent per year, then you might decide to spend the $100 today, so $100 today is worth more to you than $110 in a year. Your *discount rate* is between 10 and 15 percent.

Present value and *net present value* calculations find out today's value of a series of future cash flows. If 15 percent is the lowest interest rate you will

accept, then the present value of $115 in one year is $100 today, whereas the present value of $110 in one year is about $95 today. To calculate the present value, you take a series of anticipated cash flows, apply a discount rate to them, and find out what they are worth today. Net present value assumes that you have to make some initial investment to get those future cash flows (like depositing $100 in a bank). *Future value* is the inverse, assuming that you will receive a series of cash flows, which you will then invest at the discount rate. At the end of the periods you specify, how much will your money be worth?

Present value, net present value, and future value all assume that you know the discount rate and want to find the value of a series of cash flows. If you know the cash flows but not the discount rate, a situation that might happen if you make deposits and withdrawals to a money fund with a variable interest rate, then you need to know the internal rate of return. IRR answers the question (assuming you put in money at the beginning, expect to receive money irregularly through the life of the project, perhaps invest more money as well, and finally withdraw all your money), What is the equivalent rate of interest for a simple investment?

Table 16.1 illustrates these four functions. Assume all the cash flows take place on January 1. The Investment column represents your investment in a

	Investment	Interest	Cash Flow	PV	FV	NPV
1986	−1000		−1000	335.23		−93.02
1987	0	100	100	285.51	100.00	943.03
1988	0	110	110	218.34	225.00	974.48
1989	0	125	125	126.09	383.75	995.65
1990	1000	145	1145	0.00	586.31	0.00

Present Value:	335.23
Future Value:	586.31
Net Present Value:	−93.02
Internal Rate of Return:	11.79%

Table 16.1: PV, FV, and NPV functions

project in 1986 and your payback in 1990. The Interest column represents interest payments made to you by this investment over the years. Interest and Investment sum up to a series of Cash Flows. The PV column represents the present value, on January 1, of all the future interest payments to be made to you. The NPV column is the same, but it includes the repayment of principal and your original investment in the calculation. Use NPV if you know or are concerned with the principal amounts; use PV if you want to compare two interest streams. The FV column assumes that you will receive the interest payments and invest them; it shows how much they will be worth at a given year. All these calculations assume a 15-percent discount rate.

At the bottom of the table are the values computed by the Javelin functions for the whole time period. Instead of having to develop the present or future value series just to get the answer for the project's cash flows, the functions return a constant as their answer. Javelin's IRR function shows that investing in this project is equivalent to investing money in the bank at 11.79 percent.

An additional Javelin function, PMT, tells how much the loan payments are if you know the principal, interest rate, and number of periods for the loan. You saw this function in Chapter 14 when you constructed a loan amortization table.

The definitions of all these functions follow:

- FV (cash flow, discount rate per period, number of periods): Calculates the future value of the series of cash flows over the number of periods, if invested at the discount rate(s). The future value is a constant. If payments will be the same for each period, use the constant amount as Cash Flow; if payments will vary from period to period, use a time series with the amount of the payment for each period as Cash Flow. Javelin starts at the first period of the Cash Flow time series and uses values from it for the number of periods specified (the number of periods must be a constant). If the discount rates will be the same for each period, use a constant; if the discount rates will vary, use a time series. The payment for the first period *is* discounted, despite what some early copies of the Javelin manual say.
- IRR (cash flow series, estimated rate per period): Calculates the internal rate of return of a series of cash flows, using successive approximations. Javelin uses the estimated rate per period to begin the approximations; if you do not provide it, Javelin starts with an approximation of 10 percent.

- NPV (initial investment, cash flow series, discount rate per period): Calculates the net present value of an initial investment and the resulting series of cash flows, using the discount rates. Like FV and PV, the net present value is a constant; the discount rate may be a constant or a series. Unlike FV and PV, the cash flows *must* be a series, and the first cash flow *is* discounted by one period (Javelin assumes the series of cash flows starts in the first period).
- PMT (principal, interest rate per period, number of payments): Calculates the installment payment for a loan of the given principal, using the interest rates, over the number of payments. The arguments and the result are all constants. The interest rate given must be the rate per period.
- PV (cash flow, discount rate per period, number of periods): Calculates the present value of the series of cash flows over the number of periods, using the discount rates. The arguments work the same way as FV( ); the payment for the first period *is* discounted, despite what some early copies of the manual say.

■ *Other Present Value Calculations*

Javelin's built-in present value functions are adequate for many purposes, but when they are not, you can always construct present value calculations yourself. In this section, as an example, let's construct a model that computes a rolling present value of a series of cash flows. Each year you want to know the present value *as of that year* of the remaining flows from that point to the end of the series.

Table 16.2 shows the numbers in this calculation, and Figure 16.1 shows the formulas. Flows (the first column of numbers) is a series of cash flows running from 1987 to 1998. To get the running present value requires several steps. First you have to discount each flow back to a base year (in this case 1986) to make the amounts comparable. Then you have to compute the sum of the discounted amounts from each year to the end of the series. Finally, you undiscount the sums to make the values as of their current dates.

To make the calculation a little easier, make a series Discount Factor that for each year has the factor to apply to discount each year's numbers back to 1986, the base year. Then to get the discounted flows, divide Flows by Discount Factor. Reverse CUM, as its name suggests, calculates for each date the sum of Discounted Flows from that date to the end of the series. These

are the 1986 values of the remaining flows at each date. To determine the amounts as of their respective dates, you multiply back by Discount Factor, producing the desired PV.

	Flows	Discount Factor	Discounted Flows	Reverse CUM	Rolling PV
1987	3500.00	1.090	3211.01	65,429.38	71,318.02
1988	8400.00	1.188	7070.11	62,218.37	73,921.65
1989	7200.00	1.295	5559.72	55,148.26	71,418.59
1990	10,800.00	1.412	7650.99	49,588.54	69,998.27
1991	11,700.00	1.539	7604.20	41,937.54	64,526.11
1992	9900.00	1.677	5903.05	34,333.35	57,580.46
1993	9800.00	1.828	5360.94	28,430.30	51,971.70
1994	8300.00	1.993	4165.49	23,069.37	45,967.16
1995	7700.00	2.172	3545.29	18,903.87	41,057.20
1996	13,400.00	2.367	5660.30	15,358.58	36,359.35
1997	14,200.00	2.580	5502.97	9698.28	25,025.69
1998	11,800.00	2.813	4195.31	4195.31	11,800.00
Rate	9.00%				
PV	65,429.38				
FV	184,030.91				

Table 16.2: Numbers for rolling cash flow

```
Formulas for all variables that affect Rolling PV:

Rolling PV = Reverse CUM * Discount Factor
. Reverse CUM = Discounted Flows + NEXT(Reverse CUM);Discounted Flows
. . Discounted Flows = Flows / Discount Factor
. . . Flows = < Yearly data: 1987 - 1998 >
. . . Discount Factor = (1 + Rate)^SINCE(1986)
. . . . Rate = 0.09
```

Figure 16.1: Formulas for rolling cash flow

As a check, note that for the last year, 1998, Rolling PV is the same as Flow for the year, as you'd expect. Rolling PV for the first year is greater than the value computed by the PV function at the bottom of the listing. That's because your Rolling PV does not discount the first period, while the built-in PV function does.

The Sample Model: FUND

■ It is unlikely that you will ever use the whole complement of financial functions in a single model. To illustrate how you might do financial calculations in Javelin, here is a model that uses only one financial function (IRR) but uses other functions to do financial work. This model analyzes an investment in a fund. The question is, Given how much money you invested when, and what income you received when, how well is the investment doing? The best way to measure the success of the investment is to calculate its effective interest rate. As with most funds, the interest is variable, as is the share price. This makes it tricky to track by hand, but fairly straightforward in Javelin.

The input data consist of three data series: Share Purchase, Share Price Reported, and Dividend. Redemptions from the fund are recorded as negative purchases. Table 16.3 shows a worksheet used to enter data in this model. Because there are three input series and because the data are daily, it can be difficult to find the right date if you try to enter everything in the Table view. But with a data-entry worksheet like this one, not only is data entry

	Share Purchase	Dividend	Share Price Reported	Total Shares	Current Market Value
29 Sep 1986	893.000		11.21	893	$10,010.53
31 Oct 1986		10.966	11.62	904	$10,504.08
28 Nov 1986		2.874	11.40	907	$10,337.98
2 Jan 1987		18.483	10.99	925	$10,169.30
30 Jan 1987		4.066	12.06	929	$11,208.43

Table 16.3: Inputs for FUND model

simplified, but you have an easy way to check whether you have entered a specific transaction. Total Shares and Current Market Value are calculated variables described below.

One design decision in creating this model may not be clear: Should purchases, redemptions, and dividends be recorded in dollars or shares? In many cases it doesn't make much difference. But you should always try to figure out what the *unit of account* is in the real-life situation you are modeling, and reflect that in your model. In this case, because this fund reports dividends in shares, we chose shares as the unit of account and multiply by share price to get dollar amounts.

Figure 16.2 shows the Roots view of the IRR formula. If the share price changes in this fund, changes in Share Price Reported will make up a major component of your gains and losses. Because transactions may occur any day and you want to record the daily performance of the fund, the model is daily. But you don't want to acquire and enter daily price information for this fund. Therefore, you enter the share prices only on the days you have transactions. For all other days, you interpolate the price based on the data entered. Total Shares and Current Market Value are straightforward sums and multiplications of the input data.

The IRR function returns a constant, so it is meaningful to calculate the IRR only for the entire series. Furthermore, IRR requires a cash flow series. The cash flow for this model consists of the investments and redemptions you have made, plus any dividends received. The series Cash Flow reflects this, but it does not take into account the current market value of the investment in the fund. For IRR to give a meaningful answer, it needs to include the entire course of the investment: initial investment, intermediate cash flows (deposits, withdrawals, dividends), and liquidation of the investment. That is why the IRR formula must include a redemption for the current market value of the fund at the date for which you want the IRR, as well as all other cash flows.

```
IRR = IRR(Cash Flow, 0.01) * 365
. Cash Flow = -Current Market Value @ 1 Feb 1987; (Share Purchase; 0 + Dividend; 0) * Share Price
. . Current Market Value = Total Shares * Share Price
. . . Total Shares = CUM(Share Purchase;0) + CUM(Dividend;0)
. . . . Share Purchase = < Daily data: 29 Sept 1986 - 29 Sept 1986 >
. . . . Dividend = < Daily data: 31 October 1986 - 30 January 1987 >
. . . Share Price = INTERP(Share Price Reported); PREVIOUS(Share Price)
. . . . Share Price Reported = < Daily data: 29 Sept 1986 - 30 Jan 1987 >
```

Figure 16.2: Roots view of IRR

To get the calculation started, provide IRR with a guess of .01 in the formula. Finally, you multiply IRR by 365; since the period of Cash Flow is daily, IRR is daily. To make it yearly, multiply by 365.

Several other numbers are of interest: Figure 16.3 shows the formulas for Dollar Purchases, Net Expenses, Net Gain, and Net Income. Table 16.4 shows a summary worksheet for this model.

■ Getting IRR to Converge

If you take the IRR of a long series of payments, for example, 15 years of months, Javelin may have trouble computing the IRR and returns #ARITH rather than an answer. You can fix this by providing a second argument to IRR that is an estimate of the correct IRR value. If you have an idea of what the correct answer should be, enter that idea as the second argument. If you

```
Current Market Value = Total Shares * Share Price
Dollar Purchase = Share Purchase * Share Price
Net Expenses = CUM(Dollar Purchase); 0
Net Gain = Net Income - Net Expenses
Net Income = (CUM(Dividend); 0) + Current Market Value
```

Figure 16.3: Other formulas in FUND model

	Net Gain	Net Expenses	Net Income	Dollar Purchase	Dividend	Curnt Mkt Val	Cash Flow
Sep 1986	11.44	10,010.53	10,021.97	10,010.53		10,021.97	10,010.53
Oct 1986	504.52	10,010.53	10,515.05		10.966	10,504.08	127.42
Nov 1986	320.04	10,010.53	10,330.57		2.874	10,316.73	32.76
Dec 1986	−9.27	10,010.53	10,001.26		0.000	9987.42	0.00
Jan 1987	1234.29	10,010.53	11,244.82		22.549	11,208.43	252.16
Feb 1987	1234.29	10,010.53	11,244.82			11,208.43	−11,208.43

Table 16.4: Summary worksheet for FUND model

don't, one way to get a good estimate is to use IRR to make an approximation, using the formula

Approximate IRR = IRR(YEARLY(Flows)) / 12

This example converts the monthly series to an annual series that is only one-twelfth as long, which makes it much easier for Javelin to get an answer, then divides the annual answer by 12 to get a monthly estimate. This monthly number is not the correct answer but is in the ballpark, so you can use it as the estimate for the monthly IRR.

Depreciation

■ Javelin has functions to help you calculate depreciation using each of the three most common methods: *straight line, double-declining balance,* and *sum of the years' digits.* Each depreciation function returns a series whose values are the amount by which the asset depreciated during the period. You must specify the initial and residual values of the asset, the number of periods you wish to depreciate it for, and the first period in which to depreciate the asset. All these values must be constants.

The depreciation functions create series with the same period as the begin date you specify. That is, if you specify a year as the begin date, you get back a yearly series. This is usually what you want; if you want quarterly depreciation, create a yearly series and convert it to quarterly.

Here are the definitions of the depreciation functions:

- DEPRSL (begin value, end value, number of periods, begin date): Calculates depreciation using the straight-line method. This method also works for periods smaller than years, by giving the begin date in the period you want to use.
- DEPRDECL (begin value, end value, number of periods, begin date, factor for decline): Calculates depreciation using the declining-balance method. If you omit the factor for decline, Javelin uses 2, resulting in double-declining balance.
- DEPRSOYD (begin value, end value, number of periods, begin date): Calculates the depreciation using the sum-of-years'-digits method.

Table 16.5 illustrates depreciation of a $20,000 asset over 10 years, showing its book value for all three methods of depreciation. The columns labeled with Javelin function names use those functions to calculate the depreciation. The other columns use simple formulas that work for this particular case, as an illustration of the general method. The formulas are shown in Figure 16.4.

Straight-Line Depreciation:

	Straight Line Book Value	Straight Line Depr	DEPRSL
1986	20,000	2000	2000
1987	18,000	2000	2000
1988	16,000	2000	2000
1989	14,000	2000	2000
1990	12,000	2000	2000
1991	10,000	2000	2000
1992	8000	2000	2000
1993	6000	2000	2000
1994	4000	2000	2000
1995	2000	2000	2000

Double-Declining-Balance Depreciation:

	DDB Book Value	DDB Depr	DEPRDECL
1986	20,000	4000	4000
1987	16,000	3200	3200
1988	12,800	2560	2560
1989	10,240	2048	2048
1990	8192	1638	1638
1991	6554	1311	1311
1992	5243	1049	1049
1993	4194	839	839
1994	3355	671	671
1995	2684	2684	2684

Table 16.5: Depreciation example

Sum-of-Years'-Digits Depreciation:

	Year	SOYD Book Value	SOYD Depr	DEPRSOYD
1986	1	20,000	3636	3636
1987	2	16,364	3273	3273
1988	3	13,091	2909	2909
1989	4	10,182	2545	2545
1990	5	7636	2182	2182
1991	6	5455	1818	1818
1992	7	3636	1455	1455
1993	8	2182	1091	1091
1994	9	1091	727	727
1995	10	364	364	364

Table 16.5: Depreciation example (continued)

```
DDB Book Value = (PREVIOUS(DDB Book Value) - PREVIOUS(DDB Depr));20000
DDB Depr = IF(SINCE(1986)<9,(DDB Book Value/10) * 2,DDB Book Value)
DEPRDECL = DEPRDECL(20000,0,10,1986,2)
DEPRSL = DEPRSL(20000,0,10,1986)
DEPRSOYD = DEPRSOYD(20000,0,10,1986)
SOYD Book Value = END(1995,(PREVIOUS(SOYD Book Value) - PREVIOUS(SOYD Depr));20000)
SOYD Depr = END(1995,20000 * UNTIL(1996)/TSUM(Year))
Straight Line Book Value = (PREVIOUS(Straight Line Book Value) -
                           PREVIOUS(Straight Line Depr));20000@1986
Straight Line Depr = RANGE(1986,1995,20000/10)
Year = < Yearly data: 1986 - 1995 >
```

Figure 16.4: Formulas for Depreciation example

There is one case in which you may want to specify the starting period as something other than a year. Straight-line depreciation evenly divides the value you give over the number of periods you specify. Thus, if you have a $10,000 expense that you want to divide evenly over 6 quarters, you could say

```
Extra Expense = 10000
Extra Expense Allocated = DEPRSL(Extra Expense, 0, 6, 3 Q 1987)
```

Extra Expense Allocated would have the value $1666.66 for the six quarters beginning in the third quarter of 1987. You may find this simpler than the equivalent BEGIN function:

Extra Expense Allocated = BEGIN(3 Q 1987, Extra Expense / 6, 6)

■ *Half-Year Depreciation*

Javelin's built-in depreciation functions give you the correct answer for assets you obtain in the first half of a year, since it computes the full depreciation for the first year. For assets obtained in the second half of a year, you will probably want to use the half-year rule, which states that for the first year, you take half the depreciation that would be computed for the year, and for subsequent years you take the full amount. Table 16.6 shows a case of half-year depreciation, and Figure 16.5 has the formulas.

In the first column of Table 16.6 you see Depr DB, the depreciation for an asset obtained in the first half of 1986 that cost $10,000 and will have a residual value of $1000 and a life of ten years. If you'd obtained the asset in the second half of 1986, you'd have to use the half-year rule

	Depr DB	Half Yr DB	First DB	Rest of DB
1986	2000.00	1000.00	1000.00	
1987	1600.00	2000.00		2000.00
1988	1280.00	1555.56		1555.56
1989	1024.00	1209.88		1209.88
1990	819.20	941.02		941.02
1991	655.36	731.90		731.90
1992	524.29	569.26		569.26
1993	419.43	442.75		442.75
1994	335.54	344.36		344.36
1995	342.18	205.28		205.28

Table 16.6: Half-year depreciation

```
Formulas for all variables:

Depr DB = DEPRDECL(10000,1000,10,1986)
First DB = VALUE(DEPRDECL(10000,1000,10,1986),1986) / 2
Half YR DB = First DB@1986;Rest of DB
Rest of DB = DEPRDECL(10000-First DB,1000,10-1,1987)
```

Figure 16.5: Formulas for half-year depreciation

to compute Half Yr DB, as in the second column. You compute that by figuring First DB, the first half-year's depreciation, and Rest of DB, the other nine years' depreciation. To get First DB, compute the regular depreciation with DEPRDECL( ), select 1986 value with the VALUE( ) function (since you want this value only for the first year), and divide by two to get the half-year value. Rest of DB uses DEPRDECL( ) again to compute the depreciation for the remaining nine years. The begin value here is the original begin value less the depreciation already taken.

Although this example uses declining-balance depreciation, any of the depreciation methods will work. If you do not use the half-year rule but actually take depreciation for the fraction of the first year that the asset was in use, you can use the appropriate fraction rather than dividing by two (e.g., if you bought the asset in February, you would multiply by 11/12).

Bond Pricing and Yields

■ Javelin's bond functions are, if possible, even more specialized than its depreciation functions. Briefly, the characteristics that define a bond are:

- Price: Cost of acquiring the bond.
- Face Value: Amount the issuer will pay the holder at maturity.
- Years to Maturity: Years until the issuer will pay the holder the face value.
- Coupon Rate: Amount the issuer will pay the holder until maturity (usually semiannually), expressed as a percentage of the face value.
- Payments per Year: If specified, defines how many coupon payments per year (usually two).

- Yield: Combined return to maturity of the price of the bond, the expected coupon payments, and the payment at maturity.

Javelin provides two commonly required functions to compute information about bonds: the price you should pay for a bond assuming a particular yield, and the yield to maturity of a bond assuming a particular price. All the arguments to the bond functions must be constants, and the functions return constants. The coupon rate and yields are annual and should be entered as decimals (e.g., .05 for 5 percent). Here are Javelin's bond functions:

- BONDPRICE (yield, face value, coupon rate, years to maturity, payments per year): Calculates the price of the bond with specified information.
- BONDYTM (price, face value, coupon rate, years to maturity, payments per year): Calculates the yield to maturity of the bond with the specified information. This is essentially an internal rate of return. As with IRR, if the calculation does not converge, Javelin returns #ARITH.

Table 16.7 illustrates a bond price, yield to maturity, and the equivalent IRR calculations. Figure 16.6 shows the formulas used.

■ *Determining Numbers of Years for Bond Functions*

Unless you buy a bond on the day it is issued, there will probably be a fractional number of years from the day you buy a bond until the day the issuer redeems it. Javelin 1.0 does not support fractional years in bond functions, but Javelin 1.1 and after do. The BETWEEN( ) function lets you figure out the appropriate fraction to use. For example, assume you buy a bond on 3 Mar 1987, and it comes due on 1 Jul 1992. Then the number of years to maturity is simply

```
BETWEEN(3 Mar 87, 1 Jul 92) / 365.25
```

(The average number of days in a year is 365.25.)

Bond issuers may use special synthetic calendars with only 360 days in a year, so sometimes this formula will not give exactly the same price or yield

	Bond Price Face Value	Coupon Payments	Bond Cash Flow
1 H 1987	-760	0	-760
2 H 1987	0	40	40
1 H 1988	0	40	40
2 H 1988	0	40	40
1 H 1989	0	40	40
2 H 1989	0	40	40
1 H 1990	0	40	40
2 H 1990	0	40	40
1 H 1991	0	40	40
2 H 1991	0	40	40
1 H 1992	0	40	40
2 H 1992	0	40	40
1 H 1993	0	40	40
2 H 1993	0	40	40
1 H 1994	1000	40	1040

IRR: 13.3883%
BONDYTM: 13.3883%
BONDPRICE: 760

Table 16.7: Bond Price example

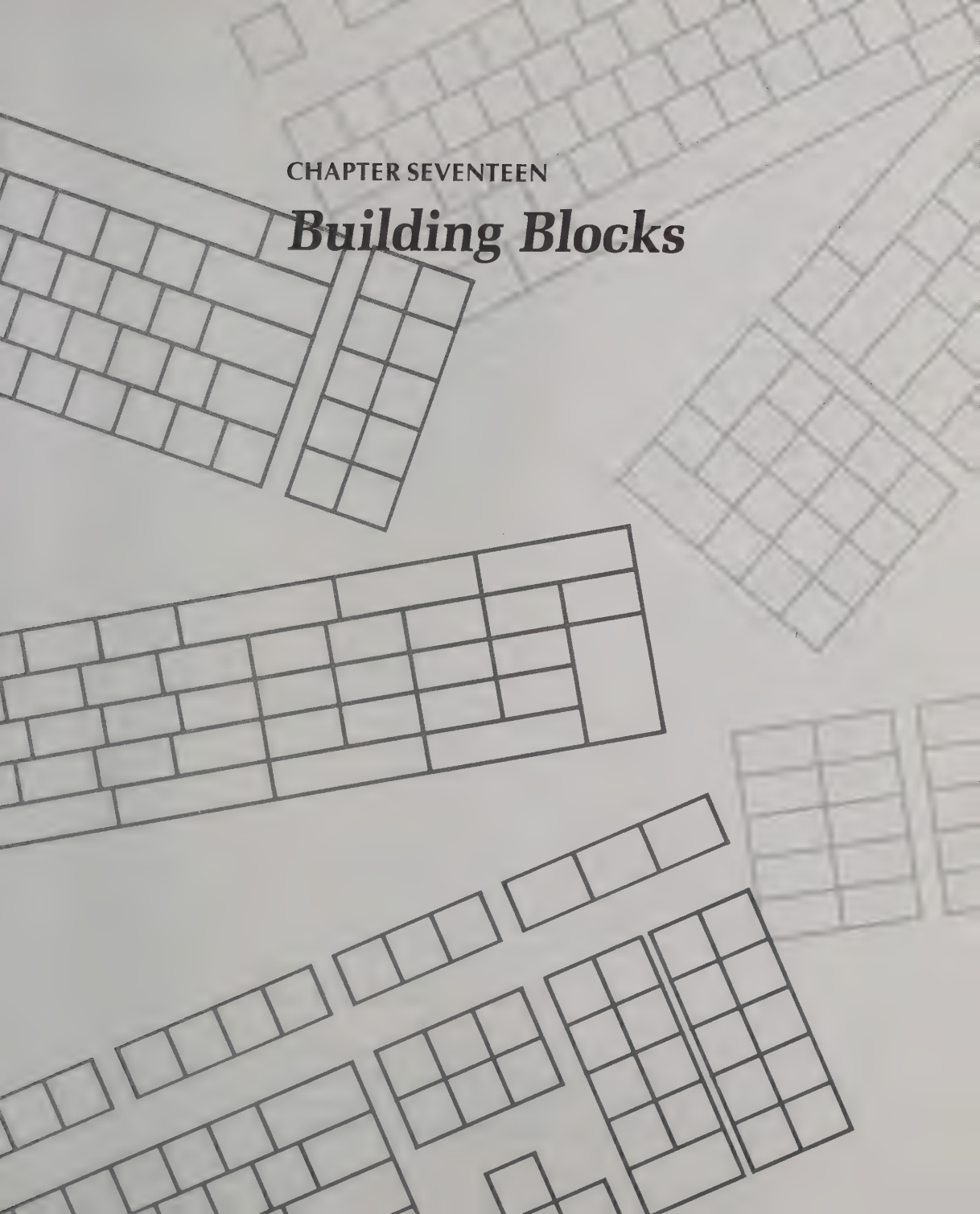
```
Bond Cash Flow = (Bond Price Face Value;0) + (Coupon Payments;0)
Bond Price Face Value = < H period data: 1 H 1987 - 1 H 1994 >
Bondprice = BONDPRICE(Bondytm,1000,0.08,7,2)
Bondytm = BONDYTM(800,1000,0.08,7,2)
Coupon Payments = < H period data: 1 H 1987 - 1 H 1994 >
IRR = IRR(Bond Cash Flow) * 2

H is a custom period defined as a half year.
```

Figure 16.6: Formulas for Bond Price example

that the issuer would compute, but the difference will probably be in the second or third decimal place.

Conclusion

■ Although Javelin's financial functions are quite specialized, even if you are not directly concerned with finance, you may find times when you need them. Present value, future value, and internal rate of return can be particularly useful for sorting out the value of investment opportunities available today, as well as for comparing the costs of different sources of funds. All these calculations could be performed without the specialized functions, but some of them would be complex and tedious. In addition, models using these financial functions will be clearer, since the functions are named descriptively.



CHAPTER SEVENTEEN

Building Blocks

Javelin's *building blocks* are like functions that allow you to express more complicated relationships between variables in your model. A building block may have one or more variables feeding into it (inputs) and can affect several variables (outputs); the building block processes the inputs to calculate the outputs. Building blocks represent relationships between variables that cannot be described using normal formulas.

You got a glimpse of building blocks in Chapter 13 in the result of / *Worksheet **R**ange **S**um in the Formulas view. Worksheet range calculations are unique in Javelin because they are the only building blocks that are not created using the command / **D**efine **B**uilding block; therefore we won't discuss them any further in this chapter. Javelin PLUS also includes *Snap In* building blocks, which allow program developers to create custom building blocks. Snap In building blocks are covered at the end of this chapter.

Introducing Javelin's Building Blocks

■ Five types of building blocks are defined in Javelin: Time Delay, Lookup Table, Curve, Import Data, and Rollup. Each performs a different set of calculations based on its inputs, and each affects its outputs differently. Just as a model may have many worksheets and graphs, it may contain several examples of any kind of building block, and you can use all five kinds in one model if you need to. The five kinds are:

- Time Delay: Input variables affect their outputs after some time delay. To indicate the time delay, you draw a bar graph in which the height of the bar at a particular period indicates what percentage of the current period's input should pass to the output at that time period. For example, Cash Receipts might be based on Monthly Billings, but with a delay of one or more months for the cash to be collected. You can use the same Time Delay building block for up to five pairs of inputs and outputs.
- Lookup Table: Using a table, such as a tax table or a discount table, this represents a relationship between pairs of input and output variables. The table of values is expressed as a base rate and a marginal rate (like $10 for the first 100 items and $2 per item thereafter). You

specify the variable(s) to be looked up and the variable(s) to contain the results, as well as a table of values.

- Curve: Gives the same kind of results as the Lookup Table, but you enter the table differently. Instead of a table of bases and marginal rates, you enter pairs of values, such as $10 for 100 items, $120 for 110 items, and $200 for 150 items. Furthermore, you enter these pairs graphically, using a curve. This method is useful when you know the shape of the relationship between the variables but not the exact values of inputs and outputs. Curves are better than lookup tables when the relationship between the inputs and outputs is smoother, not following a stepwise pattern.
- Import Data: Imports one or more variables from another Javelin model, linking the current model to the other one. You specify the name of the model to import from and the variable(s) to import.
- Rollup: Consolidates data from other models into the current model. It is a specialized version of the Import Data building block. You specify an *organizational chart,* which determines the hierarchy of the models from which the data will be copied, and the variables to be consolidated. It is used most often for corporate consolidations, in which data for each division or department are stored in a separate model.
- Import Data: Imports one or more variables from another Javelin model, linking the current model to the other one. You specify the name of the model to import from and the variable(s) to import.
- Rollup: Consolidates data from other models into the current model. It is a specialized version of the Import Data building block. You specify an *organizational chart,* which determines the hierarchy of the models from which the data will be copied, and the variables to be consolidated. It is used most often for corporate consolidations, in which data for each division or department are stored in a separate model.

These five building blocks have much in common, including the way you create, edit, and delete them. That general information is covered in this chapter, using the Time Delay building block as an example. This chapter also covers two other building blocks in detail: Lookup Table and Curve. The Import Data and Rollup building blocks are described in Chapter 18, "Transferring Information Between Models."

The Time Delay Building Block

■ In the Diagram view, a building block appears as a large box, split horizontally into top and bottom parts. The top part contains the name of the building block. The bottom part contains the details of that building block, which depend on what type of building block it is. The details appear in the form of a menu. You can choose each item in the menu to specify a part of how the building block should work.

Figure 17.1 shows how a Time Delay building block looks on the screen in the Diagram view. It is connected on the left to its input variables and on the right to its output variables. To see how all building blocks work and Time Delay in particular, you will add cash collection calculations to the books and records model that you created in Chapters 1 and 3. The result will be the building block pictured in Figure 17.1.

As you recall, the BOOKRECD model describes a business that sells books and records by mail. The amount of cash you collect is based on sales, but you don't receive payment immediately for merchandise sold. Estimate that 40 percent of the orders are prepaid, 25 percent are paid within the first month, 25 percent more are paid in the second month, and the remaining 10 percent are paid in the third month. (We'll deal with bad debts below.)

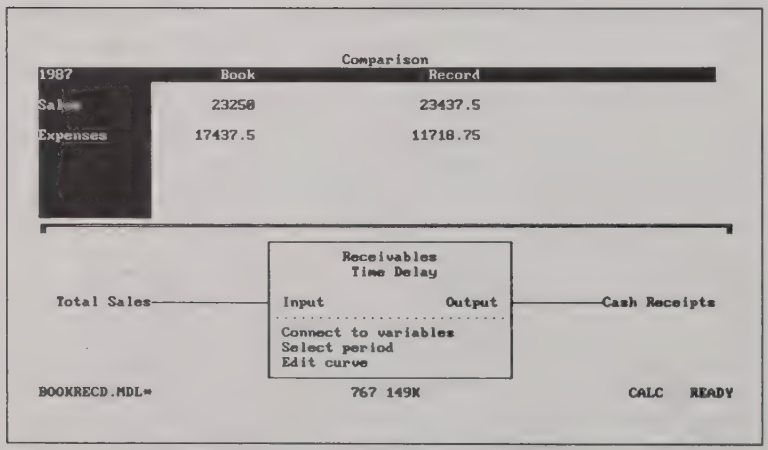


Figure 17.1: Time Delay building block in BOOKRECD model

As its name implies, in the Time Delay building block, the entire value of the input is not passed to the output variable immediately. Instead, some portion of the input variable's value is delayed and passes to the output variable only after a delay of some periods. In the first month (say, January), the results are simple: if Total Sales is $100, Cash Receipts is $40. In February, it is a little more complicated. If Total Sales is $120, Cash Receipts is $73: $48 (40 percent of $120 from February) plus $25 (25 percent of $100 from January). Table 17.1 gives sample values for the Time Delay building block, using Total Sales as the input and Cash Receipts as the output.

Time delay based on Total Sales to produce Cash Receipts

	Total Sales	Current Month × 40%		1 Mo. Prev. × 25%		2 Mos. Prev. × 25%		3 Mos. Prev. × 10%		Cash Receipts
Jan 87	$100.00	$40.00							=	$40.00
Feb 87	120.00	48.00	+	$25.00					=	73.00
Mar 87	160.00	64.00	+	30.00	+	$25.00			=	119.00
Apr 87	175.00	70.00	+	40.00	+	30.00	+	$10.00	=	150.00
May 87	180.00	72.00	+	43.75	+	40.00	+	12.00	=	167.75
Jun 87	170.00	68.00	+	45.00	+	43.75	+	16.00	=	172.75
Jul 87	160.00	64.00	+	42.50	+	45.00	+	17.50	=	169.00
Aug 87	160.00	64.00	+	40.00	+	42.50	+	18.00	=	164.50
Sep 87	190.00	76.00	+	40.00	+	40.00	+	17.00	=	173.00
Oct 87	210.00	84.00	+	47.50	+	40.00	+	16.00	=	187.50
Nov 87	230.00	92.00	+	52.50	+	47.50	+	16.00	=	208.00
Dec 87	275.00	110.00	+	57.50	+	52.50	+	19.00	=	239.00
Jan 88	190.00	76.00	+	68.75	+	57.50	+	21.00	=	223.25
Feb 88	200.00	80.00	+	47.50	+	68.75	+	23.00	=	219.25
Mar 88				50.00	+	47.50	+	27.50	=	125.00
Apr 88						50.00	+	19.00	=	69.00
May 88								20.00	=	20.00

Table 17.1: Example of Time Delay building block output

In this case, the value for Cash Receipts is the same as if its formula was

Cash Receipts = (Total Sales * 0.40) + (Previous(Total Sales, 1) * 0.25; 0) + (Previous(Total Sales, 2) * 0.25; 0) + (Previous(Total Sales, 3) * 0.10; 0)

The Time Delay building block saves you from having to type such cumbersome formulas and lets you use the same percentage structure for more than one set of variables. To make the building block work in this model, you must specify the input and output variables, the time period to use for the output variable, and the shape of the curve by which the values are spread.

Creating a Building Block

■ To use a building block in your model, you must first create it using the command / **D**efine **B**uilding block, then connect it to variables that will be its inputs and outputs, and finally enter the other specifications necessary for Javelin to do the calculations.

Load the BOOKRECD model into Javelin. Let's create the Time Delay building block to calculate the new variable Cash Receipts. Give the command / **D**efine **B**uilding block. It doesn't matter which view you are in. Since building blocks are displayed only in the Diagram view, if you are in another view when you give the command, Javelin automatically switches to the Diagram view before proceeding. Javelin needs to know which of the five kinds of building block you want to use, so select **T**ime delay.

You can have more than one Time Delay building block in the model. For instance, one could calculate collections on book sales and another could calculate record sales. That's why building blocks need names, so you can specify which one you want to work with. Enter the name

Receivables

When Javelin refers to the building block, it adds the type of building block onto the end of the name you gave it. For example, if you use F5, the Goto key, in the Diagram view to go directly to this building block, its full name will be Receivables Time Delay.

Figure 17.2 shows the building block. In the top part you see the name, Receivables Time Delay. The bottom part is a list of specifications of the

building block, which vary depending on its type. This list is another menu, on which you use the ↑ and ↓ keys to move the cursor from item to item and the ↵ key to choose the item. Unlike Javelin's other menus, you cannot select a command from a building block menu by typing its first letter. Initially, the cursor highlights the name of the building block.

The menu in the bottom half of a Time Delay building block lists the following:

- Connect to variables: Enters the names of the input and output variables.
- Select period: Enters the time period of the delay curve and the outputs.
- Edit curve: Creates or edits the curve representing the percentage of the input variable that should be assigned to each time period of the output variable.

Connecting a Building Block to Other Variables

■ A building block may have input variables; all building blocks have output variables. To specify which variables will be the inputs and the outputs, building

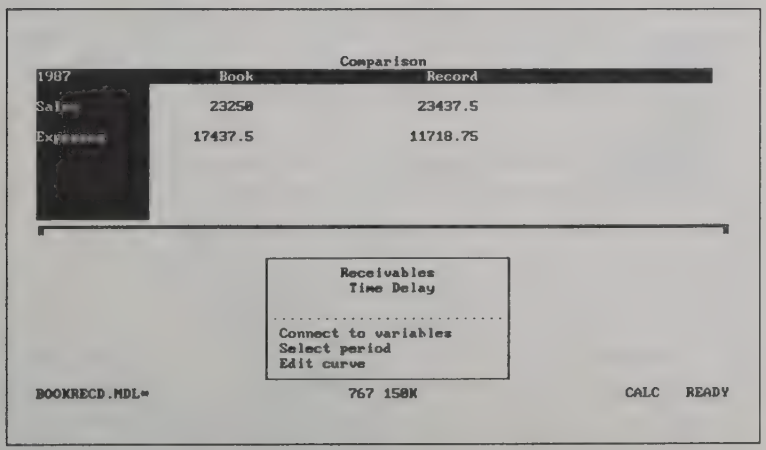


Figure 17.2: The Time Delay building block

blocks have a specification called either *Select variables* or *Connect to variables.* In either case, this selection displays a settings sheet on which you tell Javelin where the data that enter the building block should come from.

In this model, you want the building block to start with the amount of sales made, Total Sales, and end up with the amount of cash collected each month. Therefore, your input is Total Sales and your output is a new variable called Cash Receipts.

A Time Delay building block, as well as the Lookup Table and Curve building blocks, can process up to five pairs of inputs and outputs. The variable pairs don't interact with each other: each input affects only the output next to it on the Connections settings sheet. Choose Connect to variables. The Connections settings sheet appears.

In the first space under Inputs, type

Total Sales

In the first space under Outputs, type

Cash Receipts

Figure 17.3 shows the finished table of inputs and outputs for the building block.

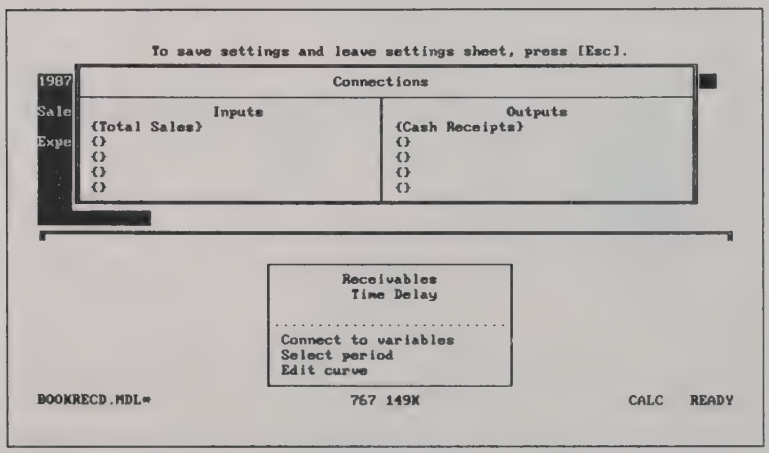


Figure 17.3: Inputs and outputs for Time Delay building block

To leave the Connections settings sheet, press Esc. You should see the Receivables Time Delay building block in the Diagram view as in Figure 17.1. The input variable on the left of the building block is Total Sales. The output variable to the right is Cash Receipts.

Selecting the Period

■ To calculate the output variable(s) based on the input variable(s), Javelin must know the period of the output variable, in this case Cash Receipts. When you define a variable using a formula, you don't have to do this because Javelin determines the period from the formula. Since Cash Receipts is defined as the output of a Time Delay building block, you have to tell Javelin explicitly what its time period is. This is important, because the amount of time you tell Javelin to delay is expressed in terms of this time period.

Normally, the time period of the building block is the same period as the input variables, as in this case with Total Sales. Even so, *you must select the period.* Otherwise, Javelin uses the model's default period. If the periods are different, Javelin converts the input variable to the output period and then puts the result through the time delay curve. For instance, if you had a monthly input but wanted a daily output, Javelin would use DAILY(input) as the input to the time delay curve; the time periods on the delay curve would be days. Having different periods for the input and output variables is rarely useful, and in the few cases where it might be, your model will be much easier to understand if you use a function like MONTHLY( ) in the input variable to make the periods match.

To make the output variable (Cash Receipts) monthly, choose **S**elect period from the building block menu. Figure 17.4 shows the available options.

Select Months, then press Esc to return to the view of the building block. You can change the period once the building block is set up, but be careful if you do; the time delay graph changes its meaning. If you change the period in this example from monthly to daily, Javelin will pass 40 percent of each day's Total Sales through on the same day, but it will delay the next 25 percent by only one *day,* the next 25 percent by two days, and the last 10 percent by three days—probably not what you want.

Editing the Time Delay Curve

■ Finally, you are ready to specify the time delays themselves, the shape of the time delay curve. Choose Edit curve, and Javelin displays a graph of the time delay curve, which you can change to reflect how you think the cash will be paid.

The time delay curve is a bar graph that specifies how much of the input variable passes through to the output variable in each period. The vertical axis indicates the percentage of the value of the input variable that should "leak through" in each period. The horizontal axis shows time periods. For each value of the input variable, Javelin divides the value among the periods of the output variable based on the percentages you indicate here. The graph itself looks and acts like a Chart view. Javelin shows the total of the percentages at the bottom of the screen.

When you first create the building block, the time delay curve shows 0 percent for each time period. For the building block to allocate each month's Total Sales amount to Cash Receipts over several months, the percentages for the months should equal 100 percent.

To create the time delay curve, use the → and ← keys to move from period to period. Once the period you want to enter is highlighted, there are

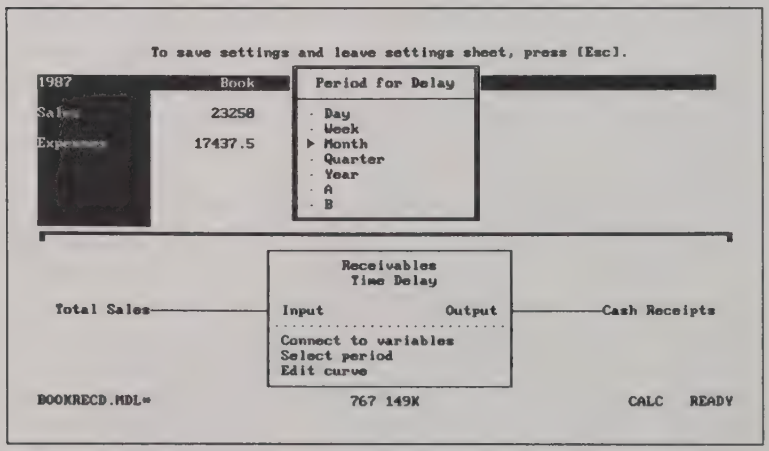


Figure 17.4: Selecting the period of the time delay

two ways to tell Javelin the percentage to allocate to it. One way is to use the ↑ and ↓ keys to adjust the height of the bar for that period until it is correct. (Javelin will show you the corresponding percentage under the graph, with the caption *n periods delay.*) The other is to type in the percentage you want as a decimal, ending your entry with ⟵. The keys you can use to enter and edit the percentages are shown in Table 17.2.

For months 0, 1, 2, and 3, respectively, enter

```
.40
.25
.25
.10
```

to create the curve. The finished curve is shown in Figure 17.5.

The total of bars doesn't have to equal 100 percent. Perhaps once the business is up and running, the real cash collection pattern is the one shown in Table 17.3. The total of the percentages comes to only 95 percent, which means that 5 percent of the sales are never paid (bad debts) or are lost to credit card fees. Enter the values in Table 17.3.

Key	Meaning
←	Move cursor left along x-axis.
→	Move cursor right along x-axis.
↑	Increase height of bar highlighted by cursor.
↓	Decrease height of bar highlighted by cursor.
PgUp	Increase height of bar highlighted by cursor by a larger amount.
PgDn	Decrease height of bar highlighted by cursor by a larger amount.
Del	Delete the bar highlighted by cursor and move following bars left to fill the gap.
Ins	Insert a bar before the bar highlighted by the cursor and move the following bars right.
Esc	Return to Diagram view of the building block.

Table 17.2: Meanings of keys when editing a time delay curve

■ *Rescaling the Delay Curve*

If you are estimating the percentages for the various periods, you may have to do some juggling to get the total of the periods to come to 100 percent. You can enter the approximate percentages so that you think the pattern looks about right, even if the total isn't exact. Then ask Javelin to do the juggling for you, by pressing F9. Javelin will adjust the entire curve proportionately so that the total equals 100 percent. In this case, when you press F9 and select **Y**es from the query menu, Javelin increases the height of each bar slightly so that they again total 100 percent.

When you have entered the time delay curve, press Esc to return to the Diagram view of the building block. Table 17.1 shows how Javelin adds up the amount received in each month. The first column shows the input to the building block, Total Sales. The next four columns show the components of Cash Receipts calculated, using Total Sales for the current month and the three previous months. Total Sales for the month indicated is multiplied by the percentage for the delay. The last column shows the resulting Cash Receipts.

Viewing, Editing, Renaming, and Deleting

■ Building blocks are visible only in the Diagram view. In the Formulas view, they appear only within the formulas for their output variables, like the / *Worksheet **R**ange **S**um in Chapter 13. But in the Diagram view, you can look at and change the input variable(s), output variable(s), and other specifications about the building block.

There are three ways to view a building block once you have created it:

- Use the Goto key, F5.
- Give the command / **V**iew **D**iagram.
- Give the command / **D**efine **B**uilding block.

Then enter the name of the building block (it's easier to use the Spell key).

You can be in any view when you use F5 or / **D**efine **B**uilding block: Javelin puts you in the Diagram view to see the building block.

It takes a fair amount of space on the screen to display a building block, even in the Diagram view. All building blocks can be shown in Javelin's default windows (half a screen high) if they only have one input/output pair. However, if they have more than one pair, you need a larger window to see them. If you try to look at a building block that is too big, you get the message *Window too small to show building block.* Give the command / **W**indow **1**.

To edit a building block, view it and then select the specification you want to change. When you are done making changes, press Esc to return to the Diagram view.

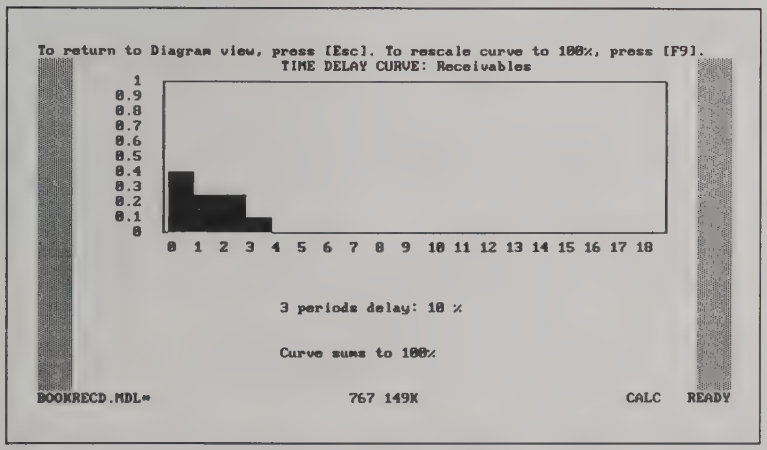


Figure 17.5: Time delay curve after editing

Month	Percentage
0	42%
1	27%
2	19%
3	7%

Table 17.3: Time delay percentages that total 95 percent

You can change the name of the building block by using the command / **D**efine **B**uilding block **N**ame change. Javelin asks for the existing name and then the new name.

To get rid of a building block, use the command / **C**lear **B**uilding block. This deletes not only the building block itself but also the output variables of the building block, unless other formulas in the model use the output variables.

Using Time Delays for Product Rollouts

■ In Chapter 9 you created the Mr. Pizza model. Your last version of that model included startup expenses for future restaurants. You can use a Time Delay building block to model the startup expenses without embedding the figures in a formula.

As you may remember, Startup Expenditures is calculated based on the number of stores opening each month. In Chapter 9 the formula was

Startup Expenditures = 50000 * NEXT(New Stores, 1) + 25000 * NEXT(New Stores, 2)

That is, two months before you open a new store, you spend $25,000, and one month before it opens you spend $50,000. With a building block it is easy to spread the delay across more than two months—a formula to do this would get very long. Let's add a cost of $5000 for the month in which the pizzeria opens.

To do this, load the MRPIZZA model and create a new building block called Startup, using the command / **D**efine **B**uilding block **T**ime delay. Because the cost of opening a pizzeria happens *before* the store opens, you need a variable to use as the input to the building block that tells you how many new stores will be opened in three months. The formula for this shifted series would be

New Stores Three Months Hence = NEXT(New Stores, 3)

The input of the Startup Time Delay building block is this shifted series and the output is Startup Expenditures. The building block replaces the formula above for Startup Expenditures. Figure 17.6 shows the Diagram view of the Startup Time Delay building block.

The curve for the building block would look like Figure 17.7. Because the input series has been shifted three months forward, a delay of 0 months is three months before the store opens, a delay of 1 month is two months before it opens, and a delay of 2 months is one month before it opens. The percentages are enormous in this building block, because the input is measured in stores and the output is measured in dollars. For one store to produce a cost of $25,000, the percentage is 2,500,000 percent (that is, 25,000 times 100 because it is expressed in percent). This is a case where your percentages do not sum to 100 percent so you should not tell Javelin to recalculate the percentages.

Table 17.4 shows the results of the building block. The first three columns show the numbers of new stores one, two, and three months previously. In the next three columns, the numbers of stores have been multiplied by the cost for that month. The last column shows the result, Startup Expenditures.

Time Delay building blocks can be used anywhere a variable affects other variables across a spread of periods.

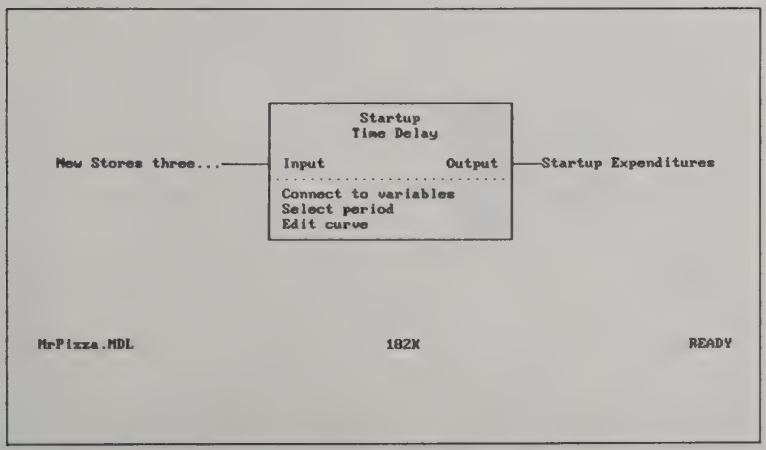


Figure 17.6: Diagram view of Startup Time Delay building block

The Lookup Table Building Block

■ The Lookup Table building block is useful for putting rate tables into your model. Two of the most popular tables are a tax table and a commission table. Both have the same structure.

A lookup table has three parts: an input interval, a base amount, and an optional rate. When Javelin looks up a value, it searches down the input interval until it finds the interval that corresponds to the input value. The result is the base amount for that interval. If there is a rate, the difference between the bottom of the interval and the input value is multiplied by the rate, and added to the base amount.

For example, Figure 17.8 shows the 1984 Federal Income Tax Schedule X as it would appear in Javelin. Following the procedure above, if the input value is $25,000, Javelin looks down the Input Over column until it gets to 23500. Since the corresponding column But not over is 28800, Javelin knows this is the range to use. The base amount for the output variable is $4115. For every dollar over $23,500 Javelin adds $.30 to the output variable. $25,000 minus $23,500 is $1500; $1500 times .30 is $450, so the output value is $4115 plus $450, or $4565.

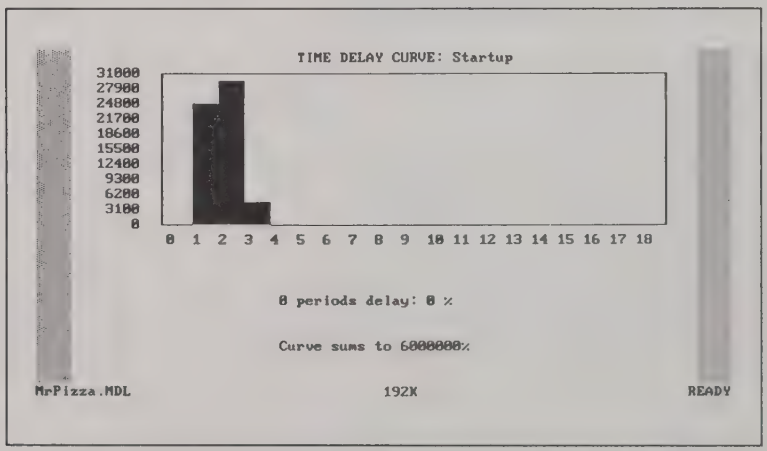


Figure 17.7: Curve for Startup Time Delay building block

Suppose you have just acquired a photocopier and have decided to sell copies. Once you decide how much to charge for copies, you can define a Lookup Table building block to calculate the charge for each copy job. Let's try three different rate schedules for copies, calculated in different ways.

For the first rate schedule, which we'll call Straight Cost, you decide that copies will cost less as more copies are made, according to these rates: 20 cents for a single copy, 10 cents a copy for two to nine copies, 9 cents a copy for 10 to 24 copies, 8 cents each for 25 to 49, 7 cents each for 50 to 99, and 6 cents each for 100 or more copies.

	New Stores Three Months Hence			Startup Expenditure				Total Startup Expen-
	1 Mo. Hence	2 Mo. Hence	3 Mo. Hence	1 Mo. Hence (* $25K)		2 Mo. Hence (* $30K)	3 Mo. Hence (* $5K)	ditures
Oct 86	0	0		0	+	0 +	=	0
Nov 86	3	0	0	75,000	+	0 +	0 =	75,000
Dec 86	3	3	0	75,000	+	90,000 +	0 =	165,000
Jan 87	3	3	3	75,000	+	90,000 +	15,000 =	180,000
Feb 87	3	3	3	75,000	+	90,000 +	15,000 =	180,000
Mar 87	4	3	3	100,000	+	90,000 +	15,000 =	205,000
Apr 87	4	4	3	100,000	+	120,000 +	15,000 =	235,000
May 87	4	4	4	100,000	+	120,000 +	20,000 =	240,000
Jun 87	4	4	4	100,000	+	120,000 +	20,000 =	240,000
Jul 87	4	4	4	100,000	+	120,000 +	20,000 =	240,000
Aug 87	0	4	4	0	+	120,000 +	20,000 =	140,000
Sep 87	0	0	4	0	+	0 +	20,000 =	20,000
Oct 87	0	0	0	0	+	0 +	0 =	0
Nov 87	0	0	0	0	+	0 +	0 =	0
Dec 87	0	0	0	0	+	0 +	0 =	0

Table 17.4: Startup Expenditures calculated by Startup Time Delay building block

To build a model that reflects this, start with an empty model and define custom time period A to be named Copies. Set the model's default time period to Copies. You need only one formula, telling how many copies you want the price for:

Copies = SINCE(1 Copies) + 1

This results in a series from 1 to 100 copies (the default time span for custom time periods is 1 to 100). All other variables in this model will be outputs of Lookup Table building blocks.

To define a Lookup Table building block, give the command / **D**efine **B**uilding block **L**ookup table. Call the building block Straight Cost. The menu in the bottom half of the Lookup Table building block lists the choices shown in Figure 17.9.

- Connect to variables: The same as for the Time Delay building block; each input variable is independently looked up and creates an answer for its associated output variable.
- Edit table: Allows you to type in the lookup table.

Just as for a Time Delay building block, connect the Straight Cost Lookup Table building block to its variables: Copies as input and Straight Cost as output. Choose Edit table to display a blank lookup table as in Figure 17.10.

When Javelin presents the blank lookup table, the table starts for values over zero, and you are expected to fill in the upper limit. If your table needs

Input Interval Over		But not over	Output Value Base amount	Rate (optional)
	-	0	0	
0	-	2300	0	0
2300	-	3400	0	0.11
3400	-	4400	121	0.12
4400	-	6500	241	0.14
6500	-	8500	535	0.15
8500	-	10800	835	0.16
10800	-	12900	1203	0.18
12900	-	15000	1581	0.2
15000	-	18200	2001	0.23
18200	-	23500	2737	0.26
23500	-	28800	4115	0.3
28800	-	34100	5705	0.34
34100	-	41500	7507	0.38
41500	-	55300	10319	0.42
55300	-	81800	16115	0.48
over 81800			28835	0.5

Figure 17.8: 1984 Federal Income Tax Table Schedule X

to include negative numbers, just move the cursor up to the entry 0.00 in the But not over column and enter the lower limit of your table. In this case, the first range is Over zero copies But not over one copy, so enter 1 in the But not over column. Javelin automatically puts 1 in the Over column for the next range. This is Javelin's way of ensuring that your lookup tables are continuous; there can be no gaps between ranges.

A second feature of the lookup table is that it must be in ascending order. If you try to type the entries out of order, Javelin responds with the message *Lookup table interval limits must go from lowest to highest.* To fix this, you can use the Ins and Del keys to insert and remove ranges from the table.

Enter the rest of the But not over range entries in the same column: 9, 24, 49, and 99. Your ranges should appear as follows:

Input Interval	
Over	**But not over**
	0
0	**1**
1	**9**
9	**24**
24	**49**
49	**99**
over 99	

Because Javelin range entries run from a base amount but not over the top amount, your range entries frequently will look a little different from

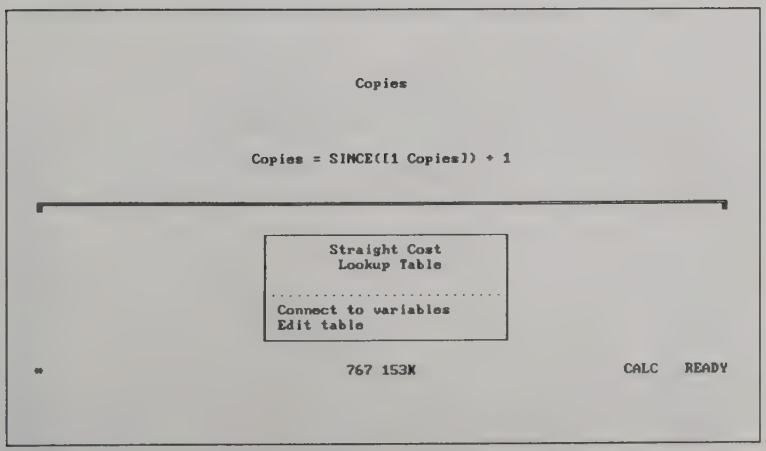


Figure 17.9: Lookup Table building block in Diagram view

how you think they should; most people would think of these ranges as 1, 2–9, 10–24, 25–49, 50–99, and 100 and over. In general, before you enter the table ranges, take a moment to make sure you are going to get the results you want.

In the right half of the lookup table, specify the output values corresponding to these input values. Move to the Base amount corresponding to over zero but not over one. Javelin divides the output into two components: a *base amount* and a *rate* per additional unit. Although the input ranges have to be continuous and in ascending order, there is no limitation on the output ranges; the base amount for the output of the second range can be less than the base amount for the first range, and the rate applied to items within a range can be negative, meaning that the output at the end of the range can be less than the output at the beginning of the range. Both these possibilities are somewhat unusual, but they should remind you that the lookup table is quite flexible.

Because Javelin sets up the output part of the table as a base amount and a rate, you may have to think a little about your table before you enter it into Javelin. For the range over 0 but not over 1 (corresponding to one copy) the charge is 20 cents. The base amount (that is, for 0 copies) should be 0 cents (making no copies is free) and each additional copy costs 20 cents. Enter the base amount as zero and the rate as .2 (20 cents).

The rest of the rates are:

- 2–9 copies: The charge is 10 cents a copy, so the base amount is 10 cents (the first copy) and each additional copy (the second through

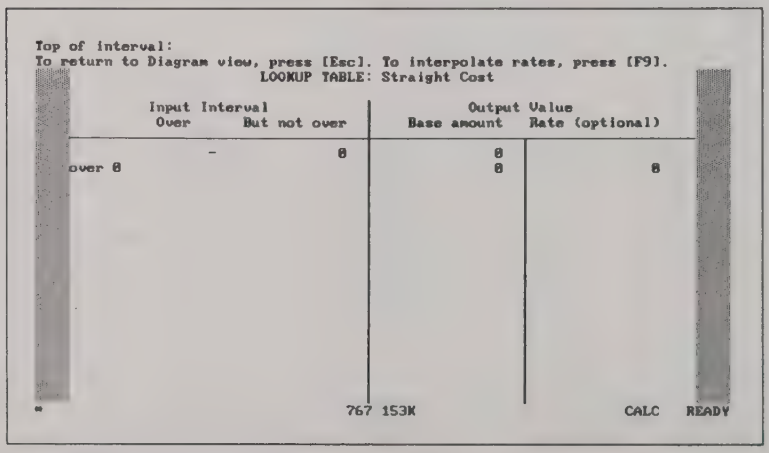


Figure 17.10: Blank lookup table

ninth) also costs 10 cents. Enter .1 for both the base amount and the rate.

- 10–24 copies: The copies are 9 cents each, so the base rate for this range is 81 cents (first nine copies at 9 cents each), and the rate is 9 cents; enter .81 for base amount and .09 for rate.
- 25–49 copies: These copies are 8 cents each, so the base amount is 24 * .08 = 1.92, and the rate is .08.
- 50–99 copies: These are 7 cents each, so the base amount is 49 * .07 = 3.43, and the rate is .07.
- Over 99 copies: At 6 cents a piece, the base rate is 99 * .06 = 5.94, and the rate is .06.

The completed lookup table should look like Figure 17.11. The building block is now complete, so exit from the lookup table with Esc.

View a worksheet to see the results. Set up a worksheet as in Figure 17.12 and recalculate the model. The default format has been set to Dollars and only one window is displayed.

A problem becomes apparent with this pricing scheme: it costs the same to make one or two copies, the same to make nine or ten copies, less to make 25 copies than 24, less to make 50 copies than 44, and less to make 100 than to make 86. It is easiest to illustrate this problem with a graph; set one up by defining a line graph of the variable Straight Cost. Where the Straight Cost line dips it is cheaper to make more copies than less (see Figure 17.13).

If you are entering a table from another source, this structure may be the way the table is intended to work. If you are creating your own table, you may want to have the charges increase smoothly. The discontinuities come from the fact that the base from one range (say $3.50 for 50 copies) plus the maximum cost of the items within that range (say the 49 copies numbered 50 to 99 times 7 cents a copy, or $3.43) add up to more than the base for the next range ($6.93 for 99 copies versus $6.00 for 100 copies). You can fix this

Input Interval			Output Value	
Over		But not over	Base amount	Rate (optional)
	-	0	0	
0	-	1	0	0.2
1	-	9	0.1	0.1
9	-	24	0.81	0.09
24	-	49	1.92	0.08
49	-	99	3.43	0.07
over 99			5.94	0.06

Figure 17.11: Straight Cost lookup table for pricing by copies

problem in one of two ways: either change the base amounts for each range or change the rates within each range. The method you select depends on which aspect of your table is important to you.

You will want to change the rates if you know that you want to charge 20 cents for one copy, 81 cents for 9 copies, $1.92 for 24 copies, $3.43 for 49 copies, and $5.94 for 99 copies. That is, if you know that the values at the *end* of the range are right, but you don't care what the rate is *within* the range, change the rates.

Javelin makes this easy: display the Straight Cost Lookup Table building block again and edit the table. Use F9, the Calc key, to ask Javelin to interpolate the rates between the bases. When you answer **Y**es to the query menu that pops up, Javelin uses the base amounts you have specified and creates new rates by interpolating between those base amounts. In the example above, Javelin will make the rate between copies 49 and 99 1/50 of the distance between the base rates ($5.94 minus $3.43, or $2.51), or 5.02 cents ($2.51 divided by 50). If you answer Yes, the new rates replace the rates that you typed in; *there is no way to get your old rates back* unless you type them in again. Always save your model before you allow Javelin to interpolate the rates, in case you don't like the answers you get. After you do the interpolation, your lookup table should look like Figure 17.14.

Look at the worksheet again (see Figure 17.15) and recalculate the model. Now the values rise evenly from 10 cents for the first copy to $6.00 for the hundredth copy. Display the graph again and you will see a smooth curve (the Interpreted Straight Cost line in Figure 17.13). But making one

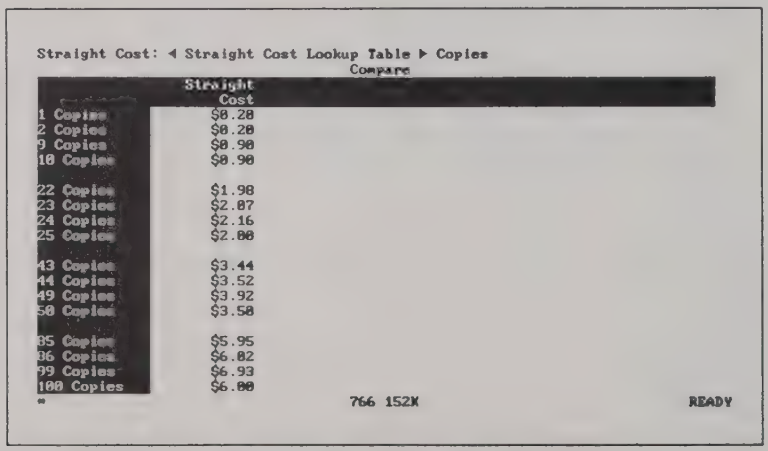


Figure 17.12: Worksheet showing prices for copy jobs

copy now costs 10 cents instead of 20 cents. When Javelin interpolated, it adjusted the rates and changed the rate for the first interval from .2 to .1. This is one reason you should check your results carefully before accepting Javelin's interpolation. The interpolation is mathematically correct, but it may not be the answer you want.

Generally you know what rate you want to charge and are less concerned about the base amounts. In other words, you want to charge 20 cents for the first copy (no matter how big the copy job is), 10 cents for copies 2 through 9, 9 cents for copies 10 through 24, and so forth. This is different from charging 9 cents a copy for jobs of 10 through 24 copies—19 cents different to be exact ($2.16 versus $2.35).

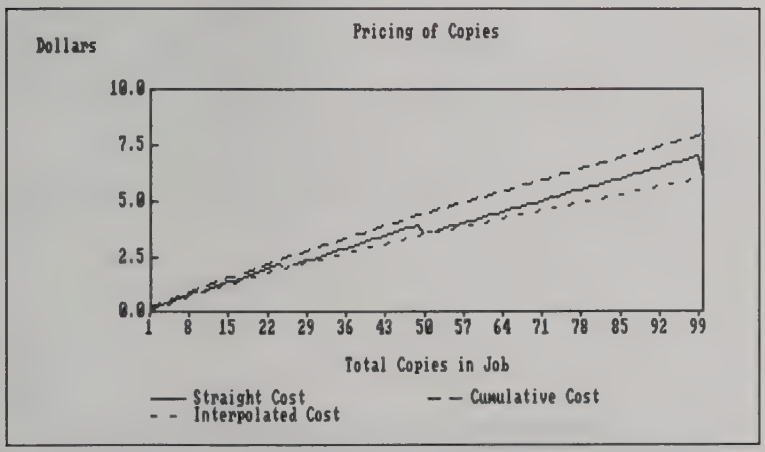


Figure 17.13: Graph showing three ways to price copy jobs

Input Interval Over		But not over	Output Value Base amount	Rate (optional)
	-	0	0	
0	-	1	0	0.1
1	-	9	0.1	0.08875
9	-	24	0.81	0.074
24	-	49	1.92	0.0604
49	-	99	3.43	0.0502
over 99			5.94	0.06

Figure 17.14: Interpolated lookup table for pricing by copies

One example of a real-life table that works this way is the tax table. The 1984 tax table in Figure 17.8 has round numbers for the percentages in the rate column, but odd figures in the base amount column. The figures in the base amount column were derived from the rates in order to make the tax curve continuous. Do the same thing for your lookup table.

Define a new Lookup Table building block called Cumulative Cost. Connect it to Copies for input and a variable called Cumulative Cost for output. Edit the lookup table.

The ranges are the same as before, so enter 1, 9, 24, 49, and 99 in the But not over column. The rates per copy are also the same, so enter .2, .1, .09, .08, .07, and .06, starting with the range 0–1. But the base amounts will be different.

Enter the base for the range over 0 but not over 1 as 0. The rate is .2, which works out to 20 cents for the first copy. The base for each subsequent range is the "top" of the previous range, that is, the value that the table will output for the maximum number of copies in the range. That makes .2 the base for the over 1 but not over 9 range.

What is the base amount for over 9 but not over 24? It's .2, plus .1 for each of the eight copies (numbers 2 through 9) in the range. The formula is

Base amount of previous range + (top of previous range – bottom of previous range) * rate for previous range

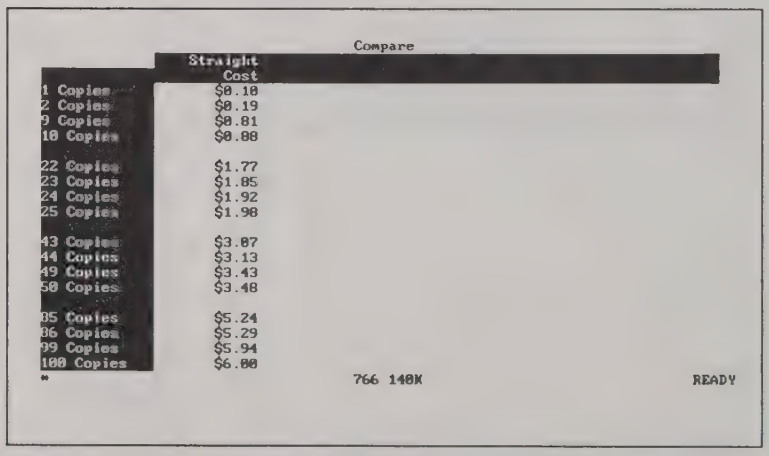


Figure 17.15: Worksheet showing Straight Cost after interpolation

Javelin allows you to enter this formula as the value for the base amount (although it stores the result of the calculation, not the formula itself). Here's what you should type to complete the table:

Range	Base Amount You Enter	Base Amount Javelin Shows	Rate
0 – 1	0	0	.2
1 – 9	0.2	0.1	.1
9 – 24	.2 + (9 – 1)*.1	1	.09
24 – 49	1 + (24 – 9)*.09	2.35	.08
49 – 99	2.35 + (49 – 24)*.08	4.35	.07
over 99 –	4.35 + (99 – 49)*.07	7.85	.06

(In Javelin PLUS you must press F9 after typing the formula and before pressing ⟵ to tell Javelin to calculate the expression.) Add a column called Cumulative Cost to your worksheet and recalculate your model. The worksheet should look like Figure 17.16.

Add Cumulative Cost to the graph and view it (see the Cumulative Cost Line in Figure 17.13). For the same number of copies, the cumulative costs are slightly higher, and there are no dips.

Copying a Building Block

■ Javelin provides no way to copy a building block within a model. However, in cases like this COPIES model, you may want to compare the output of the interpolated and uninterpolated building blocks. To avoid having to retype a building block, you can use the following procedure:

- Save the model.
- Rename the building block and, if necessary, its outputs.
- Use the command / **F**ile **I**mport **B**uilding block to import the building block you just saved back into the current model.
- Connect this building block to its inputs and outputs.

For complete information on importing building blocks from other models, see Chapter 18, "Transferring Information Between Models." Let's

try this to compare the interpolated and uninterpolated costs. Save the model as COPIES, and use / **D**efine **B**uilding block **N**ame change to rename the Straight Cost Lookup Table building block as Interpolated Cost. Using the Goto key, F5, change the output of the Interpolated Cost building block from Straight Cost to Interpolated Cost.

Use / **F**ile **I**mport **B**uilding block to import the Straight Cost building block from the COPIES model (remember, the copy on disk is still called Straight Cost). When the building block is imported, it is disconnected from its inputs and outputs, so go to the Straight Cost building block and connect it to Copies (input) and Straight Cost (output).

Finally, edit the Straight Cost lookup table so that it matches Figure 17.11 (the uninterpolated table). Only the rates need to be changed. Add Interpolated Cost to your worksheet and your graph. They should look like Figures 17.17 and 17.13. Save the model. You can use this technique to copy any type of building block.

Javelin PLUS lets you store text and dates in variables, and you can use these values in lookup tables as well. Just don't enter values for the rates.

The Sample Model: TAX

■ The classic application for lookup tables is for calculating income taxes. Figure 17.18 contains a listing of formulas to calculate U.S. federal income

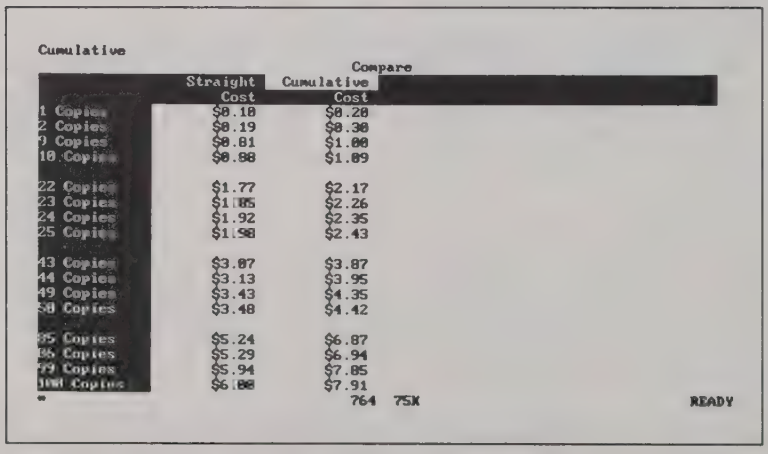


Figure 17.16: Worksheet with Straight Cost and Cumulative Cost

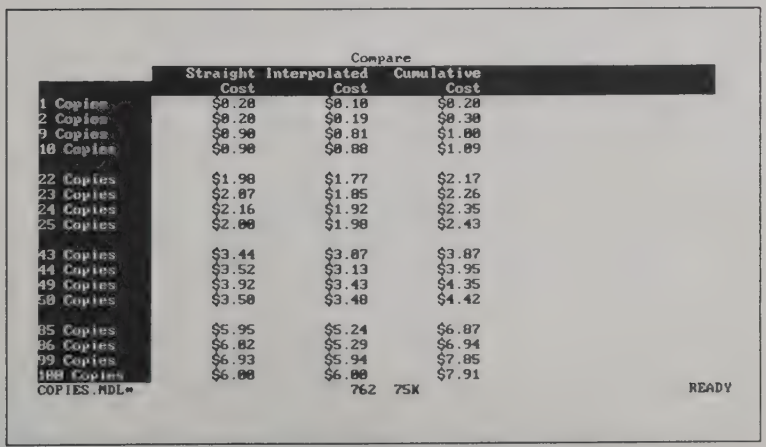
tax (the information is for 1984). All the variables are yearly, with values just for 1984. The input variables represent income, expenses, tax withheld, and number of exemptions.

Adjusted Gross Income is a worksheet range sum of Wages, Interest Income, Dividends minus Exclusion, Refunds, Business Income, and Capital Gain or Loss. The Exemptions variable is subtracted from this to get Taxable Income. Tax is calculated using a building block. The lookup table for the Schedule X Lookup Table building block is shown in Figure 17.8. The input to the building block is Taxable Income and the output is Tax. Based on Tax and Federal Income Tax Withheld, Javelin calculates Amount Owed and Refund.

The Curve Building Block

■ The Curve building block is very similar to the Lookup Table building block. In fact, almost any time you can use a lookup table, you can use a Curve building block and vice versa. But there are cases in which you will find one more convenient than the other. For instance, you could price photocopies in a Curve building block, but you would have to enter the output for each number of copies between 1 and 100—very tedious. In general, use a Lookup Table building block when

- the number of ranges is small,



Compare

	Straight Cost	Interpolated Cost	Cumulative Cost
1 Copies	$0.20	$0.10	$0.20
2 Copies	$0.20	$0.19	$0.30
9 Copies	$0.90	$0.81	$1.00
10 Copies	$0.90	$0.88	$1.09
22 Copies	$1.90	$1.77	$2.17
23 Copies	$2.07	$1.85	$2.26
24 Copies	$2.16	$1.92	$2.35
25 Copies	$2.00	$1.98	$2.43
43 Copies	$3.44	$3.07	$3.87
44 Copies	$3.52	$3.13	$3.95
49 Copies	$3.92	$3.43	$4.35
50 Copies	$3.50	$3.40	$4.42
85 Copies	$5.95	$5.24	$6.87
86 Copies	$6.02	$5.29	$6.94
99 Copies	$6.93	$5.94	$7.85
100 Copies	$6.00	$6.00	$7.91

COPIES.MDL* 762 75K READY

Figure 17.17: Worksheet showing three ways to price copy jobs

- the output values must be exact, or
- the information is already in the form of base amounts and rates.

Use a Curve building block when

- the shape of the output is complicated, so there would be too many ranges for a lookup table, or
- the exact values don't matter as much as the shape of the output.

The following example could be done in a lookup table. However, because it uses approximate figures, because the relationship is nonlinear, and because it has a fairly large number of distinct ranges, it is more convenient to use a Curve building block.

Imagine this scenario: you work for an airline and want to predict the number of customer complaints based on the percentage of load factor in a

```
Input variables (all yearly, 1984-n-1984):

Business Income
Capital Gain or Loss
Dividends
Federal Income Tax Withheld
Interest Income
No. Exemptions
Refunds
Self Employment Tax
Wages


Formulas for intermediate calculations:

Adjusted Gross Income = < Total Income 1040 US Individual Income
     Tax Return 1984 Worksheet Sum > Wages, Interest Income,
     Dividends minus exclusion, Refunds, Business Income,
     Capital Gain or Loss
Dividends minus exclusion = Dividends - 100
Exemptions = No. Exemptions * 1000
Tax = < Schedule X Lookup Table > Taxable Income
Taxable Income = Adjusted Gross Income - Exemptions
Total Tax = Tax + Self Employment Tax


Formulas for results:

Amount owed = IF (Federal Income Tax Withheld < Total Tax,
     Total Tax - Federal Income Tax Withheld, $RANGE)
Refund = IF (Federal Income Tax Withheld > Total Tax,
     Federal Income Tax Withheld - Total Tax, $RANGE)
```

Figure 17.18: Formulas for TAX model

given month (i.e., how full the planes are). Your input is Load Factor in percent and your output is Complaints per Thousand passengers. The shape of the curve is complicated by the fact that when flights get full, the airline adds extra cabin service personnel to improve the service level.

Starting with a blank model, define the Curve building block using the command / **D**efine **B**uilding block **C**urve and call it Complaints per Thousand (see Figure 17.19).

A Curve building block has three choices in its lower half:

- Connect to variables: Specifies the input and output variables. You can have up to five pairs of inputs and outputs.
- Set graph scale: Specifies the upper and lower limits of the input and the output, which will be the x- and y-axes of the curve. The defaults are 0 as the lower limit and 100 as the upper limit for both input and output.
- Edit curve: Enters the values for the curve.

Connect the Complaints Curve building block to the variable Load Factor for input and Complaints for output.

Since you enter the data graphically, choose **S**et graph scale to set the x and y limits of this graph. When you select Set graph scale your screen should look like Figure 17.20. The settings are only approximate, to help Javelin set the scale of the graph; the exact upper and lower values displayed

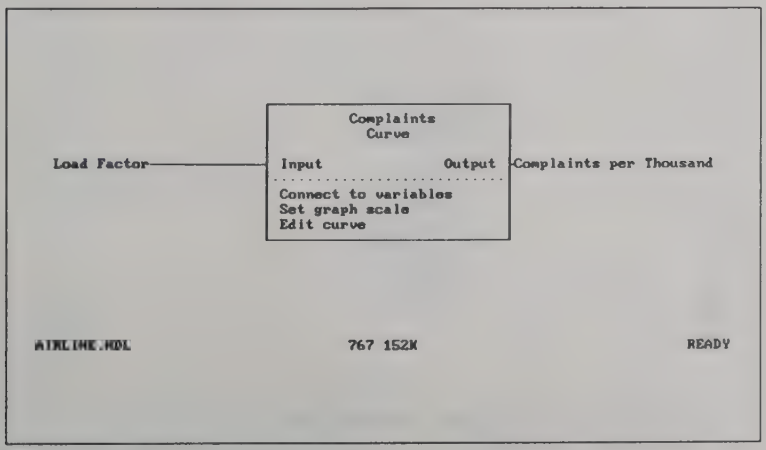


Figure 17.19: Curve building block in Diagram view

depend on the resolution of your screen as well. Note that Javelin can display and edit the curve on monochrome screens, which generally cannot display graphics, as well as on configurations with graphics capacities. Set the output high to 50 and press Esc to return to the Diagram view.

To specify the relationship between the inputs and the outputs, select **E**dit curve. A blank curve appears. You enter data on this curve just as you did on the Time Delay building block curve: ← and → select the input range, and ↑ and ↓ stretch and shrink the output bar corresponding to that input. As with time delay, if you have an exact figure in mind, you can type it in directly.

Enter a curve that looks like Figure 17.21. This curve reflects the fact that extra personnel are added when planes are 75-percent and 85-percent full. Note also that, because of the scale on the input axis, you cannot necessarily specify your input points exactly— you have to use the points Javelin selects. If precision is very important you should use the Lookup Table building block, although by trying different scales on the Set graph scale part of this building block, you may be able to set what you want. Javelin interpolates linearly between the points you specify.

The results of this building block are illustrated in Figure 17.22. The Load Factor column contains data for estimated load factors for the year; Complaints per Thousand is the output of the building block.

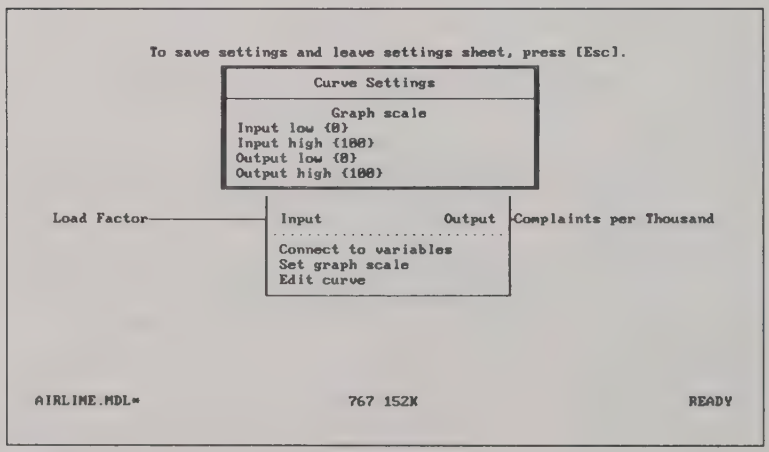


Figure 17.20: The Curve Edit scale settings sheet

Snap In Building Blocks

■ A new feature in Javelin PLUS is Snap In building blocks, which you add to Javelin as you need them. Typically, they perform specialized functions that would not be of general interest to Javelin users but that will be useful for a certain group. Javelin PLUS comes with a few examples of Snap In building blocks; others are available from Javelin Software and other companies. As more Snap Ins are developed in the future, or if you decide you need an existing one, you can buy them and "snap them in" to Javelin. The Snap Ins that come with Javelin PLUS are: What If (described below), Multiple Regression (multivariable linear regression), Histogram (see Chapter 14), and Forms Entry (data entry forms for macro-driven applications).

■ Declaring a Snap In

When you first get a Snap In building block, you must tell Javelin that it exists. Since they may be distributed separately from Javelin PLUS, Snap In building blocks are stored in separate files on your disk, similar to models.

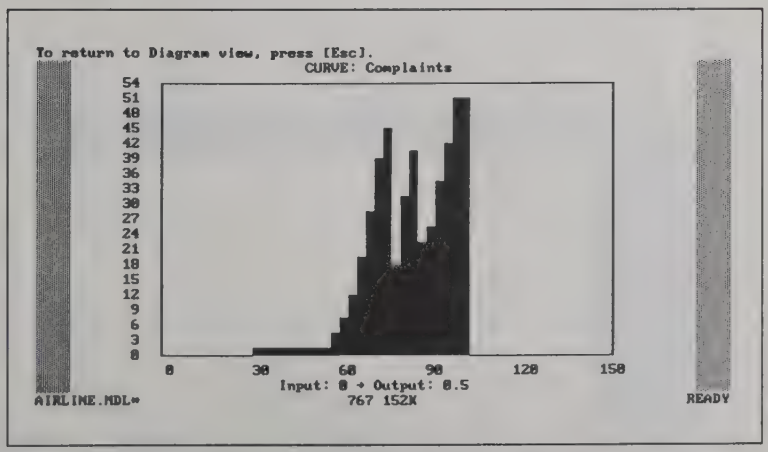


Figure 17.21: Expected Complaints as a function of Load Factor

/ **D**efine **B**uilding block **D**eclare tells Javelin about a new type of Snap In and adds its name to the list of building blocks you can define. If you have a hard disk, before declaring a snap-in you must copy it into the same subdirectory that contains the Javelin PLUS program.

When you give the command / **D**efine **B**uilding block **D**eclare, Javelin prompts you for the name of the file that contains the Snap In building block program. Once you have declared a Snap In building block type, you do not have to declare it again, *if you save the Javelin configuration.* Therefore, once you have declared the building block, give the command / **D**efine **C**onfiguration and select *Save configuration settings permanently.* Once your Snap In type is declared, you define the building blocks in much the same way you define any other.

■ *Defining a Snap In*

To use a Snap In building block, give the command / **D**efine **B**uilding block **S**nap in. Javelin asks you for the type of the building block. Use ⟶|, the Spell key, to see the possibilities and select one. Then as with any other building block, Javelin asks for the name and you enter it.

Once the building block is defined, it behaves like any other. The selections vary on each type of building block, but generally they have a way to

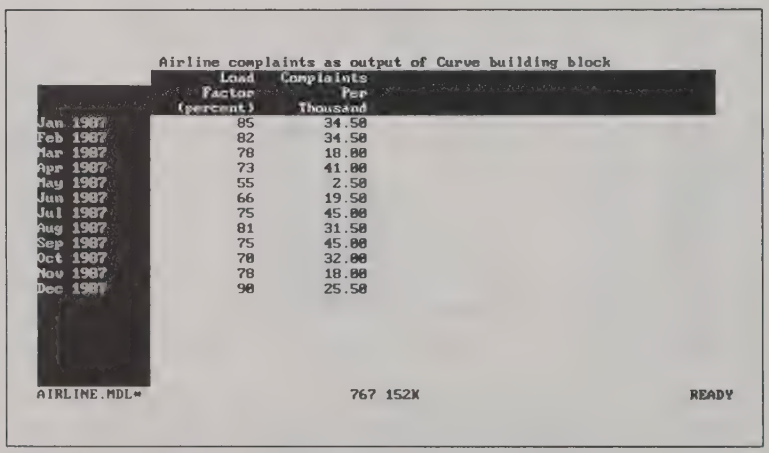


Figure 17.22: Airline Complaints as output of Curve building block

connect them to input and output variables, and selections to set the particular options for the block. In many cases, the first selection is Connect to variables, which displays the same Connections settings sheet that the Curve, Lookup Table, and Time Delay building blocks use.

The What If Snap In Building Block

One of the more useful Snap In building blocks that comes with Javelin PLUS is the What If building block. It lets you do sensitivity analysis to see what happens to your model if you change the values of some inputs. Spreadsheet users will recognize this as similar to the /Data Table 1 command in Lotus 1-2-3. Since What If is included with Javelin, we will use it to demonstrate Snap Ins.

The What If building block is contained in a file called WHATIF.JBB, which is on one of the Javelin program diskettes. If you have a hard disk, copy this file to your Javelin program subdirectory. Give the command / **D**efine **B**uilding block **D**eclare to declare it. Javelin will prompt you for the name of the file. If you don't have a hard disk, put the diskette with the building block file in Drive A:. Enter

```
whatif
```

Javelin confirms that the Snap In has been declared. If you plan to use this building block type in other models, save the Javelin configuration by using / **D**efine **C**onfiguration to display the Configuration settings sheet and selecting Save configuration settings. You will not have to declare this building block type again.

In order to perform the what if analysis, this building block needs four variables, two inputs and two outputs:

- Input Series (input): A set of input values that the What If building block plugs into the model one at a time. The series can be in any time period but is usually in a custom period called Record. Record is described in more detail in Chapter 20, "Strings and Dates."
- Test Parameter (output): A variable (constant, not a time series)into which the What If building block plugs the various values from the input series. This is where your model looks for the value being tested.

- Test Result (input): Another variable (constant) that the model should compute based on the test output. Test Result is usually a formula that refers to the Test Parameter variable.
- Output Series (output): Another series which will contain the various test results, with the time periods matched to the input series. The Test Result value is copied into this series each time What If calculates your model.

■ *Using What If on Mr. Pizza*

Recall that the Mr. Pizza model borrows and repays money to the bank, computing the maximum line of credit required to open 32 new pizzerias. The What If building block is the perfect tool to see how sensitive these plans are to changes in the interest rate. Let's use it to determine the line of credit you need for different interest rates.

Load MRPIZZA and give the command / **D**efine **B**uilding block **S**nap in to create a building block using a Snap In type. Display a list of the Snap In types that you have declared using the Spell key. Select a block type of What If and call the block Rates. Javelin creates a building block called Rates What If. The block has one selection, Connect to variables. Select it and you'll see the settings sheet in Figure 17.23.

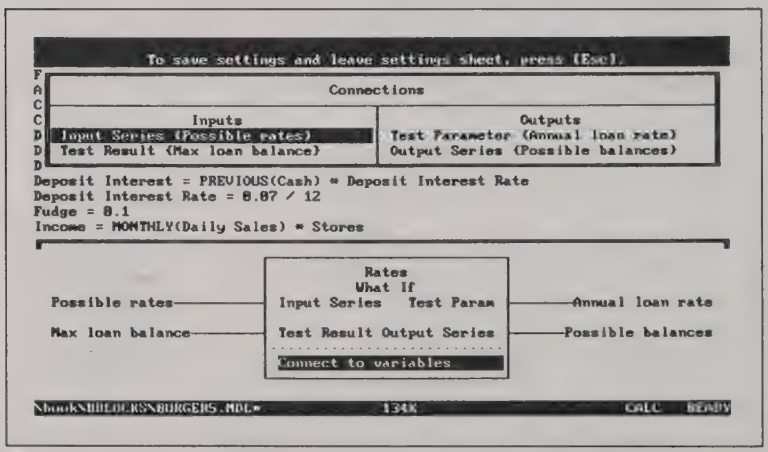


Figure 17.23: Connecting a What If building block

Connect the following variables to the What If building block:

- Input Series: Possible Rates.
- Test Parameter: Annual Loan Rate.
- Test Result: Max Loan Balance.
- Output Series: Possible Balances.

The What If building block will use each value of Possible Rates as Annual Loan Rate, calculating Max Loan Balance for each value. It will store the values for Max Loan Balance in the series Possible Balances.

The series Possible Rates is just a list of the rates you want to test, so it should be defined in terms of a nontime custom time period. Use / **D**efine **V**ariable to set the time period of Possible Rates to Record. You want to test five interest rates ranging from 7 percent to 18 percent. Display a Table view of Possible Rates and enter these values:

1 Record: .07
2 Record: .08
3 Record: .10
4 Record: .12
5 Record: .18

Then modify the formula for Loan Interest Rate to define it in terms of the Annual Loan Rate defined by the building block:

Loan Interest Rate = Annual Loan Rate / 12

The new and modified variables are listed in Figure 17.24.

When you push F9, Javelin calculates the model five times, one for each test value, and stores the five results in Possible Balances. The worksheet in Table 17.5 shows the computed balances. In this case, the line of credit required does not vary a great deal depending on the interest rate.

This example is typical of real uses of What If building blocks. Most often, you've already created a model and need to know how crucial some assumptions are. Modify the model by putting a What If building block in place of one assumption to see how the results vary. Having more than one What If building block in your model does not exactly simulate what 1-2-3 does with /Data Table 2, but it is useful. If you have more than one What If (you may have as many as you like), Javelin takes the first datum from each of the input series, copies them to their respective test values, computes the

```
Annual Loan Rate = < Rates What If > Possible Rates
Loan Interest Rate = Annual Loan Rate / 12
Possible Balances = < Rates What If > Max Loan Balance
Possible Rates = < Record period data: 1 Record - 5 Record >
```

Figure 17.24: Modified variables in Mr. Pizza model

	Possible Rates	Possible Balances
1 Record	7.00%	609,128.76
2 Record	8.00%	610,652.69
3 Record	10.00%	613,711.47
4 Record	12.00%	616,784.86
5 Record	18.00%	626,093.28

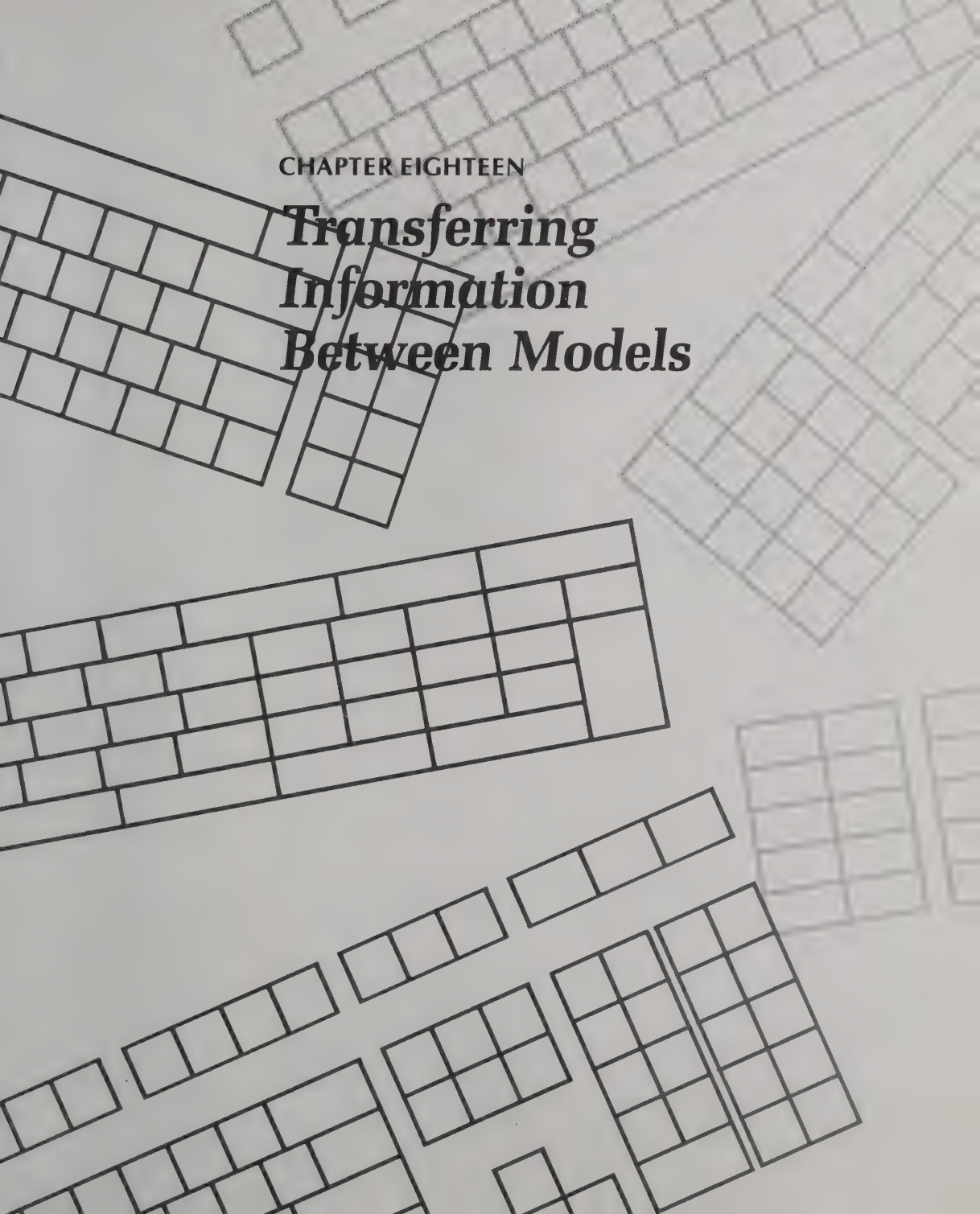
Table 17.5: Calculated maximum loan rates using What If building block

model, and copies each of the test results to their corresponding output series. It does not create a multidimensional table of output for each possible combination of inputs.

You can also use What If building blocks for goal seeking, as a quick way to plug in possible values for an unknown input to see how close they get to a desired output.

Conclusion

■ Building blocks are a unique and powerful feature of Javelin. The standard building blocks fall into two classes: those for data manipulation (Time Delay, Lookup Table, and Curve, covered in this chapter), and data import (covered in the next chapter). In addition, Snap In building blocks allow you to add custom modules to Javelin if you have unusual modeling requirements. Javelin Software maintains a list of available Snap In building blocks. Call their support line for information if you think you need one.



CHAPTER EIGHTEEN

Transferring Information Between Models

There are three general reasons to copy information from one Javelin model to another:

- Reusing data: If you want to use the same input data in several models or use the output variables of one model as input data for another, you copy input or result variables from one model to another. This extends to sharing data with other Javelin users—when several users need common information in their models or one user wants to include the results of another model.
- Consolidating: If there are models for each part of a company or project and you need to create a summary with information from all parts, you must import results from the detail models into one summary model.
- Reinventing the wheel: If you have created a formula, building block, worksheet, macro, or other component of a model and you need it in another, rather than reentering it you want to copy the component from one model to the other.

Javelin provides three ways to *import* information from one model to another:

- Import Data building block: Copies one or more variables from a model on disk into the current model.
- Rollup building block: Copies variables from a hierarchically arranged set of models and "rolls them up" into summary variables in the current model.
- / **F**ile **I**mport commands: Copy components of another model into the current model. These commands allow you to copy not just data but macros, custom time periods, and other items. The command / *Worksheet **X**port **J**avelin provides another way to copy data from one model to another.

We will start with the building blocks, using a consolidation model to illustrate how they work. Then you'll see how to use the / **F**ile **I**mport commands to "borrow" elements of existing models when setting up a new one. All these building blocks and commands are used only to copy information from one Javelin model to another. To import data from other programs, see Chapter 19, "Using Javelin with Other Programs."

The Import Data Building Block

■ In Chapter 17, you saw how to define and use a building block. The types you have used so far manipulate variables in the current model to create other variables.

The Import Data building block allows you to read in data from one or more variables from a given model, and to assign them new names when they are imported into the current model. Javelin imports the values of the variables, not their formulas.

Suppose your company does most of its work with the federal government. When evaluating your sales figures, two calculations are of interest: the company's sales growth adjusted for inflation and the percentage of the federal budget captured as revenue. The ECON model, which you developed in Chapters 8 and 12, contains the data for this analysis; we will bring these data into a new model using the Import Data building block.

Starting with an empty model, give the command / **V**iew **F**ormulas **A**ll in the bottom window to see the variables as they are defined, and / **V**iew **D**iagram in the top window. When you give the command / **D**efine **B**uilding block **I**mport data, you define a building block called Econ Import Data (see Figure 18.1). In the Diagram view, you will see two items on the building block's menu:

- Select variables: Displays a settings sheet to enter the file and variable names you want to use.
- Import values: Performs the import.

■ *Selecting Variables to Import*

When you choose **S**elect variables, Javelin displays the settings sheet shown in Figure 18.2 (in Javelin 1.1, not all the settings exist). At the top, you enter the name of the *source model* from which you want to import data. All the variables must come from the same model. You can use the Spell key to see a list of models on the disk. If the source model is not in the current directory, specify the full path name. As the file to import, enter

ECON

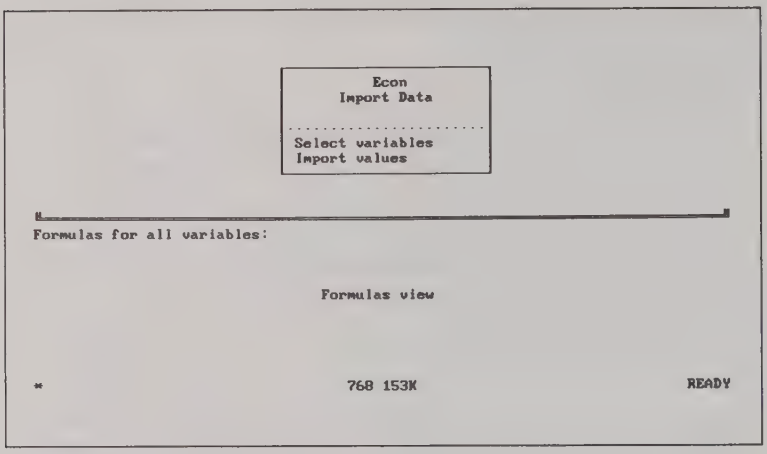


Figure 18.1: The Import Data building block

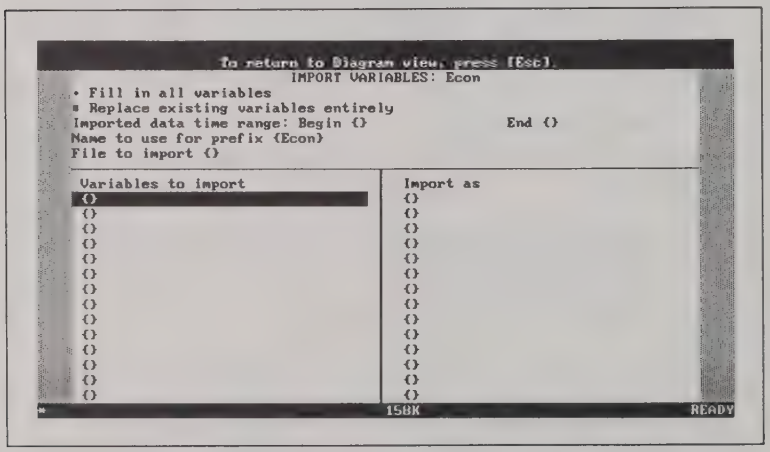


Figure 18.2: The Import Data Select variables settings sheet, Javelin PLUS

Below this is space for pairs of variable names. For each pair, the variable name on the left is the name in the source model and the name on the right is the name to use in the current model. As soon as you enter the variable name in the source model in the left column, Javelin suggests a name for the variable in the current model, adding the name of the building block to the beginning of the name; you can change this if you wish. For example, if

you enter GNP Deflator for the name in the source model, Javelin automatically makes the name in the current model Econ GNP Deflator. If you want to change the name in the current model, move the cursor onto the name in the right column and change it (use F2, the Edit key, to bring it onto the edit line).

The variable name must match the name in the source model or Javelin won't be able to find it. If you specify names that do not exist in the model, Javelin imports values for the variables it does find, and displays the message *Values imported, but some variables not found.* To figure out which variables were not imported, give the commands / **V**iew **F**ormulas **A**ll and / *Formulas **T**ime spans. Variables that were not imported will appear as *Cannot Be Calculated.*

When Javelin searches for the variable name in the model to import from, the standard Javelin rules for matching variable names apply: the words must be the same, although capitalization, extra spaces, and word order (in Javelin PLUS) don't matter. If you don't remember the variable names, put your cursor on an empty row in the left column and press ⟶| to see a list. In this model, use the Spell key to import the variable Federal Spending as well as GNP Deflator. The only limits to the number of variables you can import are the memory in your machine and the limit of 16,000 variables for all models.

■ *Improvements in Javelin PLUS*

Javelin PLUS gives you more control over importing. Two toggle settings and two value settings have been added to the top of the Select variables settings sheet.

- Import all variables: Instead of having to type the name of each variable you want to import, you can move the cursor to Import all variables. If you press ⟵|, all the variables in the source model appear in the variable list. You may then delete the ones you don't want.
- Replace previous variable entirely/Retain data outside imported time span: In Javelin 1.1, imported variables replace all the data that may have existed for the variables, including data outside the range of what's being imported. For example, suppose you had previously imported Federal Spending for 1966 through 1985; if you now import

from a model with Federal Spending for 1986 and 1987, the data 1966 through 1985 are *lost*. In Javelin PLUS this toggle setting tells Javelin that only the data imported should replace the existing data.

- Begin at/End at: Normally, a variable's entire time span is imported. If you restrict the time span by specifying begin and end dates, new data are imported only within that time span. What happens to any existing data in your model outside those dates depends on what you specified for Retain/Replace: they either remain unchanged or are deleted.
- Name to use for prefix: If you don't want the names of imported variables to begin with the building block name, you can change them as a group rather than individually. If you enter text for the setting Name to use for prefix, any variables in the import list beginning with the building block name will be changed to begin with the text you specify.

■ *Performing the Import*

Press Esc to exit from the settings sheet. The building block retreats to the Diagram view; in the Formulas view you should see your two variables defined as the output of the Econ Import Data building block. To perform the import, choose **I**mport values from the building block. Javelin copies data from the ECON model into the new variables; the ECON model is not affected. Give the command / **V**iew **T**able Econ GNP Deflator to verify that the data were loaded; note that they are in billions of dollars.

If you change the data in ECON later, those changes will *not* automatically be reflected in this new consolidation model. You must choose Import values again when you load this model. The command / **F**ile **I**mport **I**mport values (described below) is useful when you have several Import Data building blocks.

Table 18.1 shows nominal sales figures for your company, in thousands of dollars; enter them in the Table view.

To see how fast your sales really grew, you want to look at those figures in 1982 dollars. Econ GNP Deflator is a number between 1 and 100 describing what percent of a 1982 dollar a current-year dollar was worth. So define Sales 1982 Dollars as

Sales 1982 Dollars = Nominal Sales / Econ GNP Deflator * 100

Similarly, to find out what percentage of GNP your sales were, divide sales by total federal spending. You need to multiply this by one million because your sales are in thousands and the federal spending figures are in billions:

Relative to Govt = Nominal Sales / (Econ Federal Spending * 1000000)

Change the format for this variable to Percent, with four decimal places.

	Nominal Sales
1966	$97,058
1967	115,494
1968	120,260
1969	140,542
1970	160,262
1971	145,252
1972	182,442
1973	208,309
1974	244,404
1975	248,099
1976	268,466
1977	346,136
1978	387,732
1979	393,556
1980	436,926
1981	466,031
1982	471,370
1983	469,615
1984	540,566
1985	605,575

Table 18.1: Nominal sales data

Table 18.2 shows a worksheet summarizing the results. Although nominal sales increased almost fivefold, when adjusted for inflation they grew slightly less than double. Although your percentage of the federal government spending increased from .12 percent to .17 percent, this represents a decline from its maximum of .23 percent in 1978. When additional figures become available for 1985 or if the previous inflation figures are changed (as they sometimes are), all you have to do is choose Import values from the building block and the figures from the ECON model will be updated.

■ Budget Consolidation Using Import Data

The Import Data building block could also be used to create a corporate consolidation from the Budget model you created in Chapter 13. Your goal is to construct a single model that automatically consolidates the actual, variance, and percent variance figures from the budgets of your five offices.

To combine values from more than one budget, you would start with a blank model and define an Import Data building block for each detail budget model. You probably wouldn't need to import all variables from the detail models—the totals would be enough. Once you have imported all the data you want from the detail models, you can define formulas to sum them up. This could be a tedious process if you have many detail models; as usual in such cases, Javelin has a special feature to speed it up.

The Rollup Building Block

■ The Rollup building block works like the Import Data building block, except that it can import the same variables from many models and automatically total them according to a hierarchy that you define. This feature is useful for duplicating an organizational structure, consolidating geographically distributed data (as in the budget example) or making a hierarchical summary (for instance, if you had various investment fund models of the type you created in Chapter 16).

To tell Javelin how to consolidate your models, you create an organization chart (*org chart* for short). The bottom boxes on an org chart contain the model names from which you wish to import data; the higher levels contain

names that Javelin uses as prefixes to the consolidated variable names (just as it used the Import Data building block name as the prefix for the imported variables). After specifying the org chart, you specify the names of the variables to import. Unlike in the Import Data building block, in the Rollup building block you are stuck with the variable names Javelin creates for the variables you import: the name of the block in the org chart plus the variable's name in the original model.

	SALES		
	Nominal	**1982 Dollars**	**Relative to Govt**
1966	$97,058	$277,309	0.1209%
1967	115,494	321,710	0.1246%
1968	120,260	318,992	0.1201%
1969	140,542	353,121	0.1405%
1970	160,262	381,576	0.1622%
1971	145,252	327,144	0.1454%
1972	182,442	392,348	0.1724%
1973	208,309	420,826	0.1958%
1974	244,404	452,600	0.2103%
1975	248,099	418,379	0.1920%
1976	268,466	425,461	0.1970%
1977	346,136	514,318	0.2289%
1978	387,732	537,025	0.2396%
1979	393,556	500,707	0.2211%
1980	436,926	509,832	0.2100%
1981	466,031	495,778	0.1923%
1982	471,370	471,370	0.1729%
1983	469,615	452,423	0.1648%
1984	540,566	500,061	0.1727%
1985	605,575	542,144	0.1711%

Table 18.2: Sales vs. inflation and federal spending

Let's do the budget consolidation using a Rollup building block. Starting with an empty model, give the command / **V**iew **F**ormulas **A**ll in the bottom window, switch to the top window, and give the command / **D**efine **B**uilding block **R**ollup. Enter

Consolidated

as the name of the building block. This automatically becomes the name of the top element in the rollup hierarchy. The building block looks like Figure 18.3.

There are three items on the building block's menu:

- Edit org chart: Enters or edits the org chart that tells Javelin the names of the source models and how to consolidate the data.
- Select variables: Enters the names of the variables in the source models.
- Import values: Performs the rollup.

■ *Creating the Org Chart*

When you choose **E**dit org chart, Javelin displays a blank org chart. There is only one box, which will be the top box of the org chart. Each box

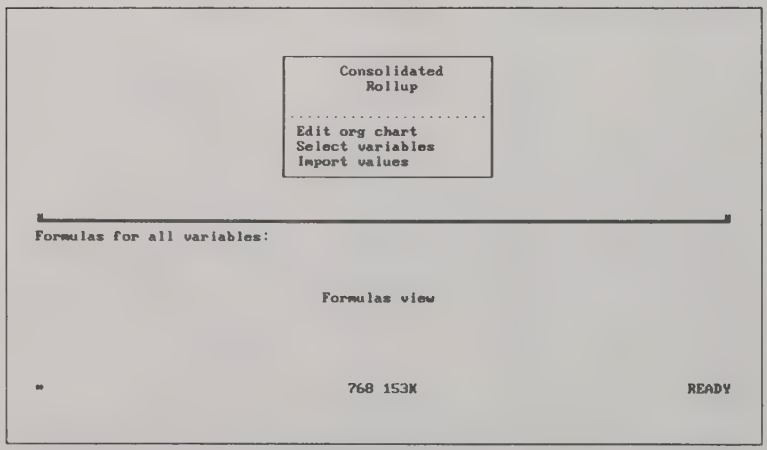


Figure 18.3: The Rollup building block

has room for a box name and a file name below it in braces. When you first display the org chart, the top box has the same name as the building block, although you can change it.

To enter a box on the org chart, move the cursor to the box above, to the left, or to the right of where you want to be and press Ins. Then press one of the following to tell Javelin where to place the new box:

- ↓: Below the box the cursor is on;
- ←: To the left of the box the cursor is on;
- →: To the right of the box the cursor is on.

Enter the name of the new box. Plan ahead when creating your org chart. Although you can insert and delete boxes, you can do so only at the ends of the tree. That is, you cannot insert a box vertically *between* two existing levels of the organization, nor can you delete a box that has boxes under it. To add or remove a level of organization, you have to recreate the entire organization below the box in question.

Assume that you have offices in Boston, New York, and Philadelphia as part of the East region, and offices in Atlanta and Dallas in the South. Let's create an org chart that will allow you to compare the actual, budget, and variance figures from the five cities and the two regions, as well as for the whole corporation. Below the Consolidated box, create boxes for the East and South regions. When the top of the org chart is the only box that exists, pressing Ins puts a box below it. Use Ins to create a box named East; the cursor is positioned in the box you create.

Create another box for South beside East, by pressing Ins. To tell Javelin where to put the box, press → and enter South as the name. Use the same technique to create two boxes below South, for Atlanta and Dallas; move up to East and create boxes for Boston, New York, and Philadelphia.

So far, you have not told Javelin where to get the data for each box. You may have noticed that only the boxes at the bottom of the org chart have braces in them; enter the file from which Javelin should import the values in those braces. As with the Import Data building block, you must enter full path names if the models aren't stored in the default directory.

Figures 18.4 and 18.5 show the halves of the org chart for your geographical consolidation. If an org chart is too large to fit on the screen, you can bring the part you want to see into view by moving the cursor toward it. To return to the Diagram view of the building block, press Esc.

■ Selecting the Variables

In addition to telling Javelin the models from which to import the data, you must also specify the variables you want. *The variables must have the same names in all models.* If you want to consolidate budget versus actual totals for each city, this information must be stored in a variable with the same name in each source model. If the variable Budget 10 Total Salaries were called Estimated 10 Total Salaries in the Dallas model, the rollup would not work. As in the Import Data building block, variables not found can be identified by looking for Cannot Be Calculated variables after giving the / **V**iew **F**ormulas **A**ll and / *Formulas **T**ime spans commands.

When you choose Select variables, Javelin displays the settings sheet shown in Figure 18.6 (in Javelin 1.1, not all the settings exist). Unlike the Import Data building block, there is only one column of variable names. You cannot assign a different name to variables as they are imported. You enter the variables using the name each has in the source models. In this example, you want to roll up the budget, actual, and variance figures for the total, and exempt and nonexempt categories in the model. You can use the Spell key to display a list of variable names from one of the models in the org chart.

For this example, here are the variables to import:

10 Actual Total Salaries
10 Budget Total Salaries

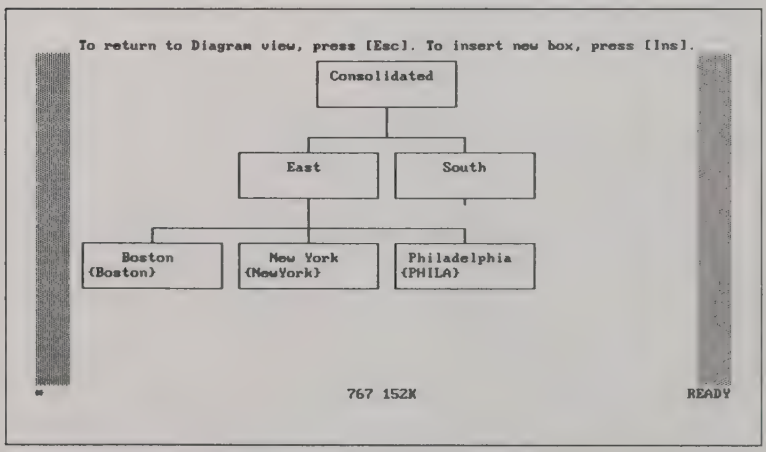


Figure 18.4: Left half of Consolidation organization chart

100 Actual Exempt Total Salaries
100 Budget Exempt Total Salaries
110 Actual NonExempt Total Salaries
110 Budget NonExempt Total Salaries
Variance Salaries 10 Total
Variance Salaries 100 Exempt Total
Variance Salaries 110 NonExempt Total

■ *Improvements in Javelin PLUS*

Several toggle settings and value settings have been added to the Select variables settings sheet in PLUS. These are at the top of the settings sheet:

- Fill in all variables: Prior to Javelin PLUS, you had to select each of these variables from the Spell key menu; now you select this setting. Javelin will read a detail model and put all its variables on the settings sheet. You can then delete the ones you don't want.
- Replace any existing variable entirely/Retain data outside imported time span: Works as described above for the Import Data building block.
- Retain leaf variables/Discard leaf variables: Described below.
- Begin at/End at: Works as described above for the Import Data building block.

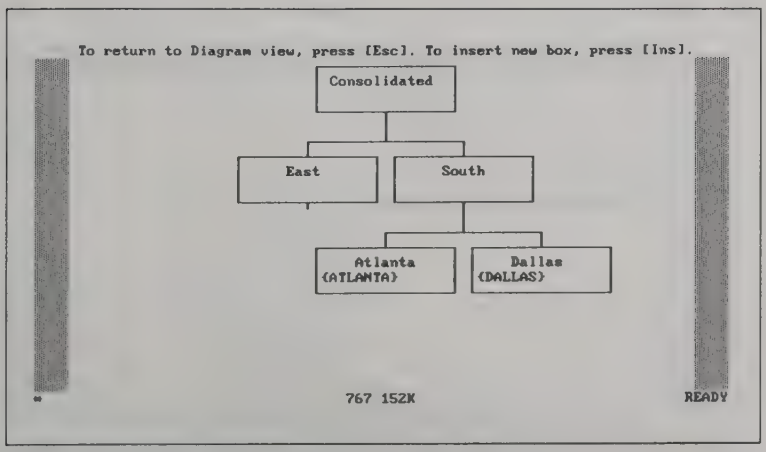


Figure 18.5: Right half of Consolidation organization chart

The Rollup building block confronts an additional problem compared to the Import Data building block: what to do if a variable being consolidated does not have data for a particular date in one of the detail models. In Javelin 1.1, if a value is missing for a date in any of the detail models, the consolidation at that date is also #NA. Javelin PLUS has a toggle setting called Ignore missing data for each variable to roll up. If you leave the setting Ignore missing data set to No (the default), it works the same as in previous versions of Javelin. If you toggle it to Yes, missing values are imported as zeros, and your consolidation doesn't result in #NA.

Javelin PLUS also helps you determine which values are missing. If values are missing from data imported for a variable, Javelin fills in the name of the model from which data was missing, next to the name of the variable.

■ *Reducing the Number of Output Variables*

As usual, you exit the settings sheet by pressing Esc. In the Diagram view you will see that the output of the building block is the list of variables you selected to be rolled up, with the name of each box added to the front. This means that the total number of variables is the number of boxes times the number of variables—a quick way to create a lot of variables in a model. The variables for the lowest-level boxes contain the values imported from the source

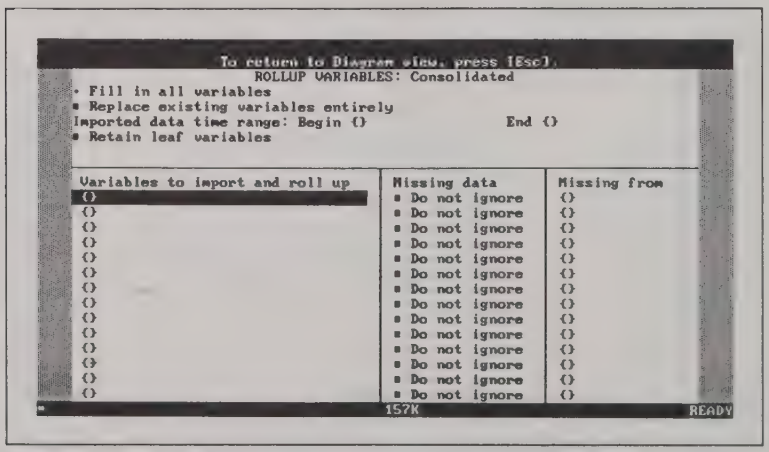


Figure 18.6: The Rollup Select variables settings sheet, Javelin PLUS

models. The variables for the other boxes contain the totals from the lowest-level boxes according to the lines on the org chart.

For example, the East variables would be the totals of the Boston, New York, and Philadelphia variables, while the South variables would be the totals of the Atlanta and Dallas variables. The Consolidated variables would be the sum of the East and South variables. The Atlanta outputs of the Rollup building block would be

Atlanta 10 Actual Total Salaries
Atlanta 10 Budget Total Salaries
Atlanta 100 Actual Exempt Total Salaries
Atlanta 100 Budget Exempt Total Salaries
Atlanta 110 Actual NonExempt Total Salaries
Atlanta 110 Budget NonExempt Total Salaries
Atlanta Variance Salaries 10 Total
Atlanta Variance Salaries 100 Exempt Total
Atlanta Variance Salaries 110 NonExempt Total

and similar for Boston, New York, Philadelphia, Dallas, East, and South (72 variables in all.)

Javelin PLUS users can avoid this plethora of variables by selecting the setting Discard leaf variables on the Select variables settings sheet. The detail variables that are summed to created the consolidation are called *leaves* of the consolidation tree. Discarding the leaves has the advantage of making your consolidation model smaller and easier to read; it has the disadvantage that you cannot see the detail data unless you load the detail model itself. In this example, if you choose to discard the leaf variables, the rollup will produce variables for East, South, and Consolidated but won't retain the variables for each city.

■ *Performing the Rollup*

As with the Import Data building block, you choose Import values to tell Javelin to perform the import and consolidation (or use the command / **F**ile **I**mport **I**mport values, as described below). Table 18.3 shows a worksheet with detail and summary results of this building block using simulated data for each of the five cities.

1987	Budget	Actual	Variance
10 Total Salaries			
Corporate	$1,677,800	$1,692,888	$15,088
East	1,052,500	1,064,199	11,699
Boston	247,600	255,665	8,065
New York	420,500	429,347	8,847
Philadelphia	384,400	379,187	(5,213)
South	625,300	628,689	3,389
Atlanta	279,500	276,136	(3,364)
Dallas	345,800	352,553	6,753
100 Total Exempt Salaries			
Corporate	976,100	981,038	4,938
East	610,000	618,982	8,982
Boston	142,500	148,077	5,577
New York	269,800	273,238	3,438
Philadelphia	197,700	197,667	(33)
South	366,100	362,056	(4,044)
Atlanta	222,000	218,674	(3,326)
Dallas	144,100	143,382	(718)
110 Total NonExempt Salaries			
Corporate	701,700	711,850	10,150
East	442,500	445,217	2,717
Boston	105,100	107,588	2,488
New York	150,700	156,109	5,409
Philadelphia	186,700	181,520	(5,180)
South	259,200	266,633	7,433
Atlanta	57,500	57,462	(38)
Dallas	201,700	209,171	7,471

Table 18.3: Summary of budget consolidation

Commands to Import Components of Models

■ The two building blocks you've seen can import only the current values of variables. Sometimes it would be useful to import other parts of a model. This is what the / **F**ile **I**mport commands are for.

Each of these commands prompts you to enter the file name of the source model, may ask you for other information, and then copies the information from the source model into the current model. Be sure to use the Spell key to find out what the items are called in the source model, since it can be hard to remember exact names.

Here are the choices on the File Import options menu:

- Javelin model: Combines the entire source model into the current model. If the source model contains variables, worksheets, graphs, or macros that have the same name as existing elements of the current model, Javelin asks whether to replace the existing items. Custom time period definitions are *not* imported from the source model.
- Worksheet: Combines the specified worksheet from the Javelin source model into the current model, including worksheet borders, text and labels, exception cells, and worksheet range calculations. No variables are imported. The imported worksheet replaces the worksheet with the same name in the current model, if there is one.
- Variable: Combines the specified variable from the source model into the current model, including notes associated with it and settings from its Variable settings sheet. For an input variable, Javelin imports its values. For a calculated variable, the formula is imported. Note that the formula won't work unless the model contains the other variables that the formula refers to. The imported variable replaces the variable with the same name in the current model, if there is one.
- Macro: Combines the specified macro from the source model into the current model. The imported macro replaces the macro with the same name in the current model, if there is one.

- Building block: Combines the specified building block from the source model into the current model. Building blocks with inputs won't work unless the model contains variables with the names of the inputs. The imported building block replaces the building block with the same name in the current model, if there is one.
- Time period: Combines the specified custom time period from the source model into the current model. Even if you have renamed the custom period in the source model, you must use the original name (A, B, or Record) here. The imported custom time period replaces the custom period with the same name in the current model (A replaces A, B replaces B, and Record replaces Record), even if you have changed the names. It is a bad idea to import a custom time period into a model after you have begun to use it, since this changes the period of the existing variables that use the custom time period.
- Import values: Imports the values specified in all Import Data and Rollup building blocks in the model. This is the same as selecting Import values for each building block.

■ *Copying Worksheets, Macros, and Building Blocks*

A handy trick is to use / **F**ile **I**mport to copy items that Javelin does not let you copy otherwise. For example, say you have a worksheet called Revenues, and you'd like to make a worksheet called Expenses that is almost the same as the Revenues worksheet. You could use / *Worksheet **C**opy to copy material from one to the other, but that does not automatically copy the structure of the borders, column widths, worksheet formats, exception cells, or range operations.

Here's a way to copy the entire worksheet:

- Save your model.
- Use / **D**efine **W**orksheet to rename the worksheet Revenues to Expenses. Now you have an Expenses worksheet, but the Revenues worksheet is gone.
- Use / **F**ile **I**mport **W**orksheet to import the Revenues worksheet from your saved model. Now you have both Revenues and Expenses worksheets.

You can use the same method to copy a macro or a building block using / **F**ile **I**mport **M**acro or / **F**ile **I**mport **B**uilding block.

Exporting a Worksheet as a New Model

■ A command added in Javelin 1.1 copies information from the current model into a new model on disk. The command / *Worksheet **X**port **J**avelin creates a new model on disk, which contains the worksheet that is displayed when you give this command, including its borders, labels, and exception cells. The new model also contains the values for all variables displayed on the worksheet. The variables are saved as input variables with the values current when you give the command. Notes associated with the variables as well as model, variable, and worksheet formats also are saved.

This command allows you to export all the information that makes up a worksheet from a model, without the formulas and other variables that go into them. The current model is not affected.

Conclusion

■ Javelin makes it easy to create large models. By setting up variables on a worksheet, you can easily create two parallel sets of variables with different names but the same formulas and relationship to each other. Therefore it's tempting to set up one large model, with different aspects of the model distinguished by their variable names.

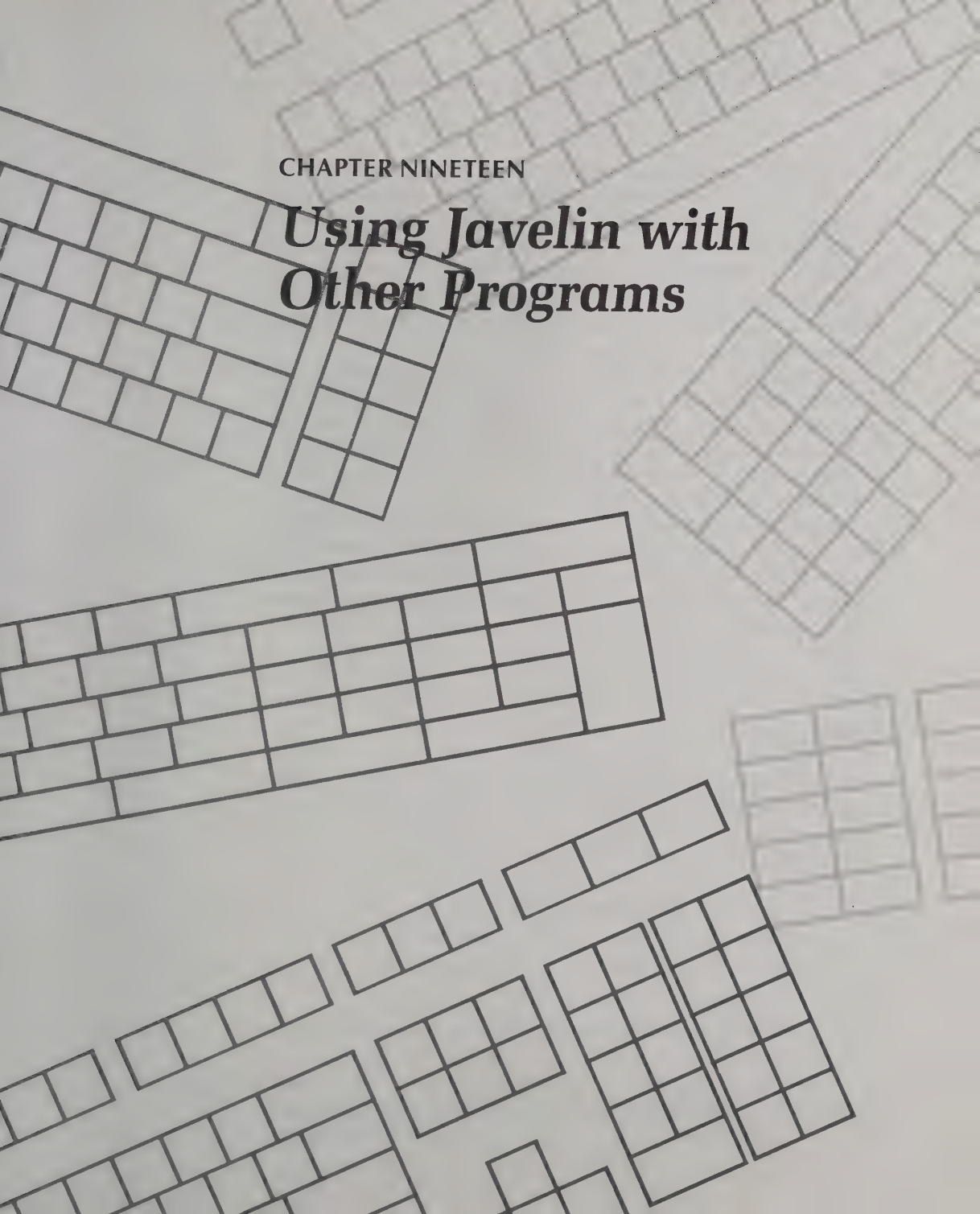
In the example used in this chapter, you could put all the budget, actual, and variance figures into one model and distinguish them by their variable names. In general, this is a bad idea. We have seen models with 500 variables, consisting of 50 formulas duplicated 10 times, but we recommend using 10 different models instead. If you ever find a problem in the 500 formula version, it will be a headache to fix. It is much easier to look at a small model, in which the lists of formulas are not as long and errors are easier to find.

Once you have broken your large model into a number of small ones, Import Data or Rollup building blocks make the consolidations easy to

develop. A warning applies to both building blocks: Javelin imports data from the models specified in these building blocks according to their values the last time each source model was recalculated. If you update data in a source model and save it without recalculating, your imports and rollups bring in the last data that was calculated, not reflecting the data entered since the last recalculation. Avoid this by always recalculating your models before saving them.

The Import Data building block can also be very useful in offices where several Javelin users are connected on a network. If you have data that one person is responsible for but several people use, the "owner" can put a model containing the data in a public place on the network. The other users can then include building blocks within their models to import that data. Every time they choose Import values, they will get the latest version of the data.

You can also create a "library" of useful features to use in models. Because Javelin lets you import custom time periods, macros, building blocks, and worksheets, if you use similar ones in many models, you can import them easily. One of your first thoughts while building a new model should be, What can I borrow from other models I have? This makes the model building process much faster, and as you develop fancier tools, makes your models more sophisticated.



CHAPTER NINETEEN

Using Javelin with Other Programs

There are two reasons you may want to transfer information into Javelin. You may be converting existing analyses to Javelin, or as part of ongoing work, you may need to use information in your Javelin models that comes from files created by another program. For example, your data may originate in a spreadsheet or database program, or may come from a mainframe or time-sharing service. Other programs store information in a format different from that of a Javelin model. Copying the information from a file into a Javelin model is called importing.

Conversely, you may want to use the results of a Javelin model in another program, for example, you may want to include figures from your worksheet in a word-processing document or transmit them to a mainframe. Word-processing programs and mainframe programs can't read Javelin model files directly. Copying the information from a Javelin model to a file in another format is called exporting.

Javelin contains commands for both purposes; all importing and exporting in Javelin is done in the Worksheet view, using commands on the Worksheet star menu. In this chapter, we will review Javelin's facilities for importing and exporting data. Then we will talk about procedures and tricks for converting a 1-2-3 spreadsheet into a Javelin model.

Data Formats

■ Before using Javelin's import and export commands, you should know about the *data formats* commonly used among PC programs, that is, the formats of the data files that programs use. Here are the common data formats and the file name extensions commonly used to specify them:

- MDL: Javelin model (all versions of Javelin).
- WKS: Lotus 1-2-3 version 1 and 1A worksheet.
- WK1: Lotus 1-2-3 version 2 and 2.01 worksheet.
- WRK: Lotus Symphony document.
- DIF: Data Interchange Format, used by many statistics and graphics programs.
- SYLK: SYmbolic LinK information, a data interchange format defined by Microsoft Corporation for use among programs. The file extension is SYL.

- ASCII Text: ASCII text file with lines of information separated by carriage returns. (ASCII is the American Standard Code for Information Interchange, the coding system used for characters in all computers except IBM mainframes.) Each line contains one piece of text information. Many file extensions are used, but when importing these files into Javelin, the file extension must be PRN.
- ASCII Delimited (also called *comma-separated variables*): ASCII text file with lines of information separated by carriage returns. Each line may contain several data items, separated by commas; all textual (non-numeric) information is surrounded by quotation marks. When importing these files into Javelin, the file extension must be PRN.

Normally, you need not be concerned with the specifics of how these formats work; just make sure you tell the exporting and importing programs to use the same formats. However, you should know how two of these formats look—ASCII Text and ASCII Delimited—because you can use them to transfer data to and from almost any program, even one that does not support the other special data formats listed above.

For example, suppose you want to create a page of text to be printed as part of your worksheet. You could type it in your word-processing program, save it as an ASCII file (most word processors can do this), and import it into a Javelin worksheet. If you have a great deal of data to enter into Javelin and a word-processing service is available, you could have them type your data, save it as ASCII files, and import it into Javelin. To do this, you must type the information in the proper format, either ASCII Text or ASCII Delimited.

■ *ASCII Text Format*

Using ASCII Text format, you enter one piece of information per line. When you import the file into Javelin, each line appears as one label in a worksheet. For example, a file containing the following:

```
Coke          404213
Pepsi         362277
Dr. Pepper     89481
```

will be imported by Javelin as labels on the worksheet, so the numbers can't be used in formulas.

■ ASCII Delimited Format

Using ASCII Delimited format, you can enter several pieces of information on each line, separated by commas. Any text information must be enclosed in quotation marks or it will be ignored when Javelin imports it. An ASCII Delimited file that looks like this:

```
"Coke", 404213
"Pepsi", 362277
"Dr. Pepper", 89481
```

can be imported into Javelin as two separate columns on a worksheet, one of labels and the second of numbers.

Any program that can write data to a disk file should be able to write data that Javelin can read. By setting up your report or document properly, you can use ASCII Text or ASCII Delimited to bring that information into Javelin.

Importing Data

■ Javelin imports data from other programs only in the Worksheet view. The imported data are placed in body cells on the worksheet. In Javelin 1.1, the data start on the first row in the leftmost body column, *regardless of where the cursor is when you tell Javelin to import the data.* In Javelin PLUS, the data start where your cursor is. The information in the border row and column of the worksheet identifies the variables and time periods into which the data are placed. If you import a file with nothing in the worksheet borders, all the data are imported as labels on the worksheet. You can then enter border information to identify the data as variables.

Javelin's ability to read different file formats has improved with every release. Here are the selections available on the / **W**orksheet **I**mport file menu and the version of Javelin in which they first appeared:

- Numbers & text: (Javelin 1.1; called Text in Javelin 1.0.) Imports an ASCII Delimited file, converting each line into a row of entries on the worksheet. Text in quotation marks is stored as a label on the

worksheet. Numbers are stored as input values on the worksheet. Any text not in quotes is discarded.

- Labels: (Javelin 1.1.) Imports an ASCII Text file, converting each line into a label on the worksheet.
- WKS/WK1/WRK: (Called WKS in Javelin 1.0.) Imports a Lotus spreadsheet file from 1-2-3 or Symphony.
- DIF: (Javelin PLUS.) Imports a DIF file, a format supported by a number of financial, statistics, and graphics programs.
- SYLK: (Javelin PLUS.) Imports a SYLK file, as created by Microsoft Multiplan and other programs.

To incorporate imported information into your model, you must enter variable names and dates into the border cells of the worksheet, so that Javelin knows how to store the contents of each cell. It is easier to enter the variable names and dates into the borders after the import, since you can see where the information appeared on the worksheet. However, it is faster and more memory-efficient to set up the borders first. To import data without knowing what variables they are associated with, Javelin must set up a temporary storage area for each piece of data, which takes more storage and processing time than reading the values directly into the storage area for variables.

■ *Importing ASCII Files*

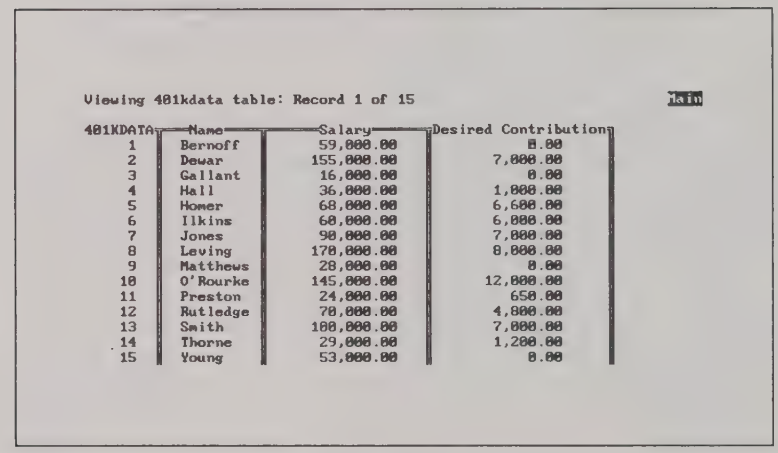
Database programs like dBASE II and III, Paradox, and PC-File can export database information as ASCII Delimited files. You can read such files into Javelin using / *Worksheet **I**mport file **N**umbers & text.

In Chapter 10 you created a database of employees, salaries, and contributions to a 401K plan. Figure 19.1 shows how this database might look in Ansa Corporation's program Paradox. Figure 19.2 shows how it might look in Ashton-Tate's dBASE III. Both programs can create an ASCII Delimited file from their database, which is shown in Figure 19.3. Figure 19.4 shows a Javelin worksheet into which this file has been imported.

If you have a file that contains only labels, with one label per line, you can import it using / *Worksheet **I**mport file **L**abels. In this case, your lines do not have to be enclosed in double quotes, but Javelin does not distinguish between numbers and text. Depending on the format of your input file, this may or may not be useful. When possible, use ASCII Delimited.

■ *Importing Lotus Spreadsheet Files*

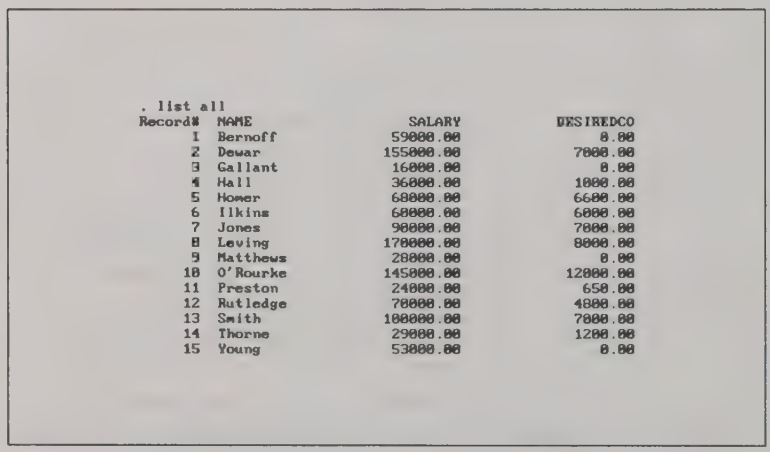
When Javelin imports a Lotus spreadsheet, it places the contents of each cell into the corresponding cell on the worksheet. Javelin 1.1 starts by placing the contents of the A1 cell from the spreadsheet in the upper-left body cell of the worksheet and imports the rest of the cells accordingly. Javelin PLUS starts by placing the A1 cell from the spreadsheet in the cell



Viewing 401kdata table: Record 1 of 15 Main

401KDATA	Name	Salary	Desired Contributions
1	Bernoff	59,000.00	0.00
2	Dewar	155,000.00	7,000.00
3	Gallant	16,000.00	0.00
4	Hall	36,000.00	1,000.00
5	Homer	68,000.00	6,600.00
6	Ilkins	60,000.00	6,000.00
7	Jones	90,000.00	7,000.00
8	Leving	170,000.00	8,000.00
9	Matthews	28,000.00	0.00
10	O'Rourke	145,000.00	12,000.00
11	Preston	24,000.00	650.00
12	Rutledge	70,000.00	4,000.00
13	Smith	100,000.00	7,000.00
14	Thorne	29,000.00	1,200.00
15	Young	53,000.00	0.00

Figure 19.1: Data for 401K model in Paradox



. list all

Record#	NAME	SALARY	DESIREDCO
1	Bernoff	59000.00	0.00
2	Dewar	155000.00	7000.00
3	Gallant	16000.00	0.00
4	Hall	36000.00	1000.00
5	Homer	68000.00	6600.00
6	Ilkins	60000.00	6000.00
7	Jones	90000.00	7000.00
8	Leving	170000.00	8000.00
9	Matthews	28000.00	0.00
10	O'Rourke	145000.00	12000.00
11	Preston	24000.00	650.00
12	Rutledge	70000.00	4000.00
13	Smith	100000.00	7000.00
14	Thorne	29000.00	1200.00
15	Young	53000.00	0.00

Figure 19.2: Data for 401K model in dBASE III

your cursor is in when you give the import command, and imports the rest of the spreadsheet accordingly.

Here is how Javelin imports the cells in a Lotus spreadsheet into a Javelin worksheet:

- If the spreadsheet cell is blank, the Javelin worksheet cell is blank.
- If the spreadsheet cell contains a label, the Javelin worksheet cell contains the label.
- If the spreadsheet cell contains a number, the Javelin worksheet cell contains the number.
- If the spreadsheet cell contains a formula, whether a numeric or string formula, the Javelin worksheet cell contains the current value of the formula.
- If the spreadsheet cell contains @ERR, the Javelin worksheet cell contains #ERROR.
- If the spreadsheet cell contains @NA, the Javelin worksheet cell contains #NA.

Javelin does not import formulas from the Lotus spreadsheet. Because Lotus formulas refer to cells rather than to variables, Javelin has no way to retain the formulas behind the values. It imports the current value of the formula when you give the import command.

Here is the general procedure for importing a Lotus spreadsheet:

- Run 1-2-3 (or Symphony) and retrieve the spreadsheet.

```
"Bernoff",59000.00,0.00
"Dewar",155000.00,7000.00
"Gallant",16000.00,0.00
"Hall",36000.00,1000.00
"Homer",68000.00,6600.00
"Ilkins",60000.00,6000.00
"Jones",90000.00,7000.00
"Leving",170000.00,8000.00
"Matthews",28000.00,0.00
"O'Rourke",145000.00,12000.00
"Preston",24000.00,650.00
"Rutledge",70000.00,4800.00
"Smith",100000.00,7000.00
"Thorne",29000.00,1200.00
"Young",53000.00,0.00
```

Figure 19.3: ASCII Delimited file produced by Paradox and dBASE III

- Determine what the variables are, what you want to call them in Javelin, and which are input variables versus calculated variables. Write down the formulas for the calculated variables.
- If your spreadsheet contains several independent sections, break them up into smaller spreadsheets by using the 1-2-3 command / File Xtract. It is easier and faster to import small worksheets than large ones. Since your Javelin model can have several worksheets, you can divide up your work more logically. Each spreadsheet you import should have one set of borders. Although you need only to import the input variables, it's a good idea to import *all* the variables, so you can check the Javelin results against the spreadsheet results.
- Run Javelin and load the model into which you want to import the spreadsheet.
- For each spreadsheet, view a new worksheet and set up the borders to match the way the data in the spreadsheet are laid out (see below for time-saving ways to do this). This is not strictly necessary, but it will make your import up to 50 percent faster. If you are importing very large spreadsheets, you may have to do this in order for them to fit in Javelin. Don't try to import several spreadsheets into one Javelin worksheet. Since Javelin allows you set up as many worksheets as you need, you can create separate worksheets with the proper borders to import each spreadsheet.

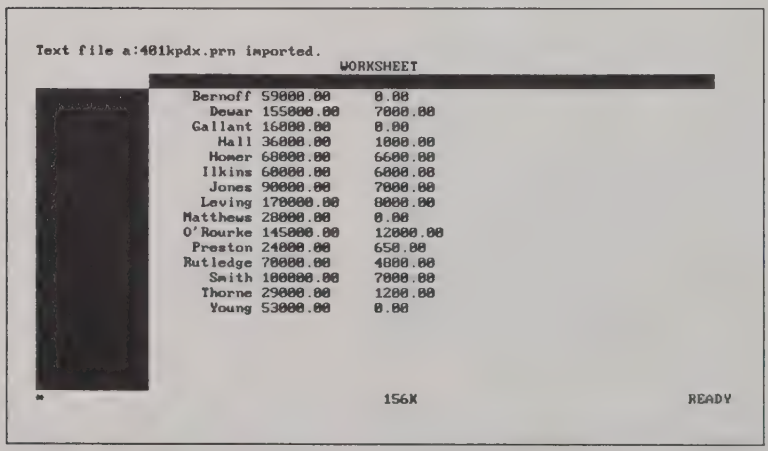


Figure 19.4: Worksheet with 401K data from ASCII Delimited file

- Import the spreadsheet into the Javelin worksheet. If the borders are wrong, clear the worksheet and try again.
- Delete any extra cells, rows, or columns that contain unnecessary labels.
- Columns that contained calculated variables were imported as numbers. Beside each of these columns, insert a new column and enter the formula that should give the same result. Recalculate the model and compare the calculated values to the imported values; adjust the formula until the answers match. Because of the way 1-2-3 does arithmetic, you may find that your Javelin and 1-2-3 answers differ by some very small number (less than 0.000001). This is because Javelin calculates more accurately than 1-2-3.

If the borders in the Lotus spreadsheet are almost the same as the variable names you want in Javelin, you can avoid typing them again. When you import your spreadsheet, the labels appear on the body of the worksheet. Using the command / *Worksheet **D**ata copy, you can copy those labels into the worksheet border.

As soon as you fill in both borders (above and to the left of a cell), you have defined a variable. If you have several rows of border labels, expand the top border to fit them (/ *Worksheet **B**order **E**xpand) and copy them *before* you insert anything in the left border. Otherwise Javelin will define a set of variables using only the first row of labels and then define another set of variables with the names in the expanded border. You will have to delete one set of variables.

If your 1-2-3 spreadsheet is small and you are only importing it once into Javelin, you do not have to set up the borders in advance, but doing so makes the import faster and uses much less memory. However, it requires more work on your part. If your spreadsheet is big, it is important to set up the borders beforehand. Here is the easiest way to do this:

- Run 1-2-3 and save a copy of your spreadsheet that contains just the top and side labels (use 1-2-3's command / Range Erase to remove the data).
- Save another copy of the spreadsheet that contains just the data (use / Worksheet Delete to remove the rows and columns containing the border labels).
- Import the label spreadsheet and copy the labels into the Javelin worksheet borders.

- Delete the labels from the body cells of the worksheet.
- Import the data spreadsheet.

■ *Importing Dates from a Lotus Spreadsheet*

1-2-3 and Javelin deal with dates in very different ways. 1-2-3 (and Symphony) store them as numbers that are displayed as dates, while Javelin considers numbers and dates to be different types of values. Therefore, you should convert 1-2-3 dates to text dates *before* importing them into Javelin as labels. Once in Javelin, you can convert these labels into Javelin dates.

If you have dates in your 1-2-3 or Symphony spreadsheet, you probably want them to end up in the border of your Javelin worksheet, as dates. (In Javelin PLUS you may want them to end up as dates in a variable; the procedure outlined below works in that case too.) Since 1-2-3 stores dates as the number of days since January 2, 1900, these numbers will be imported into Javelin rather than the dates they represent (they are in the 30,000 range for dates in the 1980s).

If your dates cover a continuous period, this is no problem. Just use Javelin's command / *Worksheet **B**order **D**ates to enter the range of dates you want in the border. But if your dates are not continuous, you may find the following procedure useful for bringing dates from 1-2-3 to Javelin.

All your dates must be in the same column of the spreadsheet. If the dates are in a row, use 1-2-3's command / Range Transpose to put them in a column.

- Load your spreadsheet into 1-2-3 or Symphony.
- Go to the column containing your dates. Make sure they are all displayed in format Date-1 (DD-MMM-YY).
- Give the 1-2-3 command / Print File, specify Option Other Unformatted, Margin Top 0, and Margin Left 0, and print this column of dates to a file.
- Still in 1-2-3, insert a new column beside the column that has the dates.
- In the new column, move to the cell beside which you started printing.
- Use 1-2-3's command / File Import Text to bring in the dates you just printed. They should line up exactly with the original dates in the neighboring column.

- Save the spreadsheet.
- Run Javelin and load your model.
- Use / *Worksheet **I**mport **W**ks to bring the spreadsheet into Javelin.
- Enter or copy variable names into the top border of the worksheet.
- Use / *Worksheet **D**ata copy to copy the dates to the worksheet's left border. They appear as labels and look similar to Javelin dates, but they have dashes rather than spaces between the three parts of the date (for example, 25–Dec–88 rather than 25 Dec 88).
- Define the macro listed in Figure 19.5. Position the cursor on the first date to be converted and run the macro (press Alt-D if you called the macro D for Dates). When the macro stops, the dashes have been removed from the labels so Javelin can interpret them as dates.

■ *Importing the FUND Model from 1-2-3*

Let's convert two 1-2-3 spreadsheets into Javelin models; both are similar to models you have used in earlier chapters: a 1-2-3 version of the FUND model from Chapter 16 and a version of the checkbook model from Chapter 10.

Table 19.1 shows the 1-2-3 version of the FUND model. Briefly, it is set up as follows:

Column	**Description:**
A	Dates, in the form @DATE(YY, MM, DD).
B, C	Input Data.
D	Total Shares. + D5 + B6, or previous total shares plus shares purchased.
E	Input Data.
F	Interpolated Share Price. @IF(E5 > 0, E5, I5 + (K5 / (L5 + K5)) * (J5 – I5)). If there is a Share Price Reported (E5), use it. Otherwise, to the last reported price (I5) add the portion of the difference between the last reported price (I5) and the next reported price (J5) corresponding to the number of days (K5) that have passed since the last price was reported.
G	Current Market Value. + D5 * F5, Total Shares times Share Price.

H	Cash Flow. (B5 – C5) * F5, Share Purchases minus dividends, times share price; at February 1, 1987, assume you receive the current market value.
A1	Internal Rate of Return. @IRR(0.2 / 365, H5..H130) * 365, which is the IRR of the daily Cash Flow series (in column H) multiplied by 365. Assume that the annual rate will be around 20 percent.
I, J, K, L	As their headings indicate; used to interpolate the share price.

Figure 19.6 shows this spreadsheet imported into an empty Javelin worksheet. This import took about 125 seconds and 60K of memory for 130 rows and 12 columns of data.

A better way to import this spreadsheet is to split the 1-2-3 spreadsheet into two before the import. One contains only the labels in the top four rows of the spreadsheet; the other contains the formulas and data. In Figure 19.7, we imported the first spreadsheet, expanded the borders, and copied the labels to the top border. Then we used / *Worksheet **B**order **D**ates to fill in 20 Sep 86 to 1 Feb 87 in the left border. Finally, we imported just the data spreadsheet into the worksheet. This took only 65 seconds and 17K of memory.

```
Macro d:

@REM Macro assumes cursor is on a cell containing a 1-2-3 date displayed
@REM in 1-2-3's DATE-1 format.  Macro converts it to a Javelin date.  It
@REM assumes that the cell below the end of the list is blank.


@LABEL Loop          @REM Tells us where to come back to the beginning.
@F2                  @REM Edit the 1-2-3 text.
@DEL                 @REM Delete the label prefix.
@IF @ERROR @THEN     @REM Delete will cause error if cell is empty.
    @EXIT            @REM      Exit from loop if so.
@ENDIF
[                    @REM VERSION 2 ONLY: ADD THE DATE BRACKET.
@CTRLRIGHT @LEFT     @REM Position on the hyphen between day and month.
@DEL                 @REM Delete it.
@CTRLRIGHT @BKSP     @REM Delete the hyphen between month and year.
@END @BKSP ]         @REM VERSION 2 ONLY: ADD DATE BRACKET.
@ENTER               @REM Enter changes
@DOWN                @REM to next date
@JUMP Loop           @REM and do it till we get an error.
```

Figure 19.5: Macro for converting Lotus dates to Javelin dates

	A	B	C	D	E	F
1	IRR of Cash Flows: 32.82%					
2					Share	
3		Share		Total	Price	Share
4	Date	Purchase	Dividend	Shares	Reported	Price
5	29-Sep-86	893.00		893	11.21	11.21
6	30-Sep-86			893		11.22
7	01-Oct-86			893		11.24
8	02-Oct-86			893		11.25
	. . .					
36	30-Oct-86			893		11.61
37	31-Oct-86		10.966	893	11.62	11.62
38	01-Nov-86			893		11.61
	. . .					
127	29-Jan-87			893		12.02
128	30-Jan-87		4.066	893	12.06	12.06
129	31-Jan-87			893		12.06
130	01-Feb-87			893		12.06

	G	H	I	J	K	L
1					Days	Days
2	Current				Since	Until
3	Market	Cash	Previous	Next	Previous	Next
4	Value	Flow	Price	Price	Price	Price
5	10010.53	10010.53	11.21	11.21	0	0
6	10021.97	0	11.21	11.62	1	31
7	10033.41	0	11.21	11.62	2	30
8	10044.85	0	11.21	11.62	3	29
36	10365.21	0	11.21	11.62	31	1
37	10376.66	–127.424	11.62	11.62	0	0
38	10369.64	0	11.62	11.4	1	27

Table 19.1: 1-2-3 spreadsheet evaluating investment fund performance

However, getting the data into Javelin is only half the battle: there are no formulas behind any of these numbers. The 1-2-3 formulas and their Javelin equivalents are as follows:

1-2-3 Column	Javelin Equivalent
A	Dates, in the form @DATE(YY, MM, DD), replaced by Javelin border.
B, C	Input Data.
D	Total Shares. + D5 + B6, PREVIOUS(Total Shares; 0) + Share Purchase.
E	Input Data.
F	Interpolated Share Price. @IF(E5 > 0, E5, I5 + (K5 / (L5 + K5)) * (J5 – I5)), INTERP(Share Price Reported).
G	Current Market Value. + D5 * F5, Total Shares * Share Price.
H	Cash Flow. (B5 – C5) * F5, (Current Market Value @ 1 Feb 87; (Share Purchase – Dividend) * Share Price).
A1	IRR. @IRR(0.2 / 365, H5..H130) * 365, IRR(Cash Flow, 0.2 / 365) * 365.
I, J, K, L	Not needed because of Javelin's INTERP function.

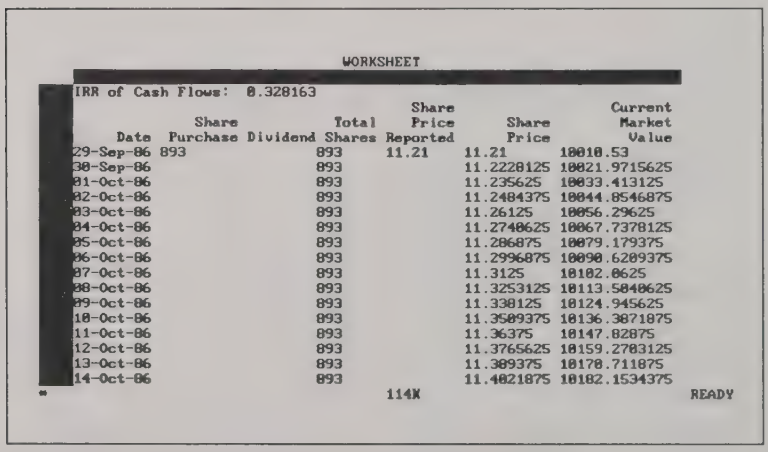


Figure 19.6: 1-2-3 fund spreadsheet imported into worksheet

In general, you should not try mechanically to translate your 1-2-3 spreadsheet into a Javelin model. For simple formulas, the conversion is straightforward, but for more complicated formulas, like the interpolated share price and the cash flow, you should rethink your formulas to make sure they take full advantage of Javelin's features. Figure 19.8 lists the formulas for the converted model; Figure 19.9 shows a formatted worksheet including the internal rate of return of the fund.

■ *Importing the Checkbook Model from 1-2-3*

Table 19.2 shows the checkbook as laid out in Lotus 1-2-3. The column titled Dates contains the dates converted to labels (see tips above).

We imported the spreadsheet into a blank Javelin worksheet. Figure 19.10 shows the worksheet with the top border expanded and the label information from 1-2-3 copied into it. We have deleted the rows in the worksheet that contained the border labels. Note that since "Check/Dep" is not a valid part of a variable name, Javelin has added double quotes before putting it in the top border (variable names can't contain slashes). Any information defined under this column will have the name Number. Check that your 1-2-3 labels are valid Javelin variable names before passing this point. In this case, replace "Check/Dep" with Trans (so your variable is called Trans Number) by using the command / *Worksheet **N**ame change.

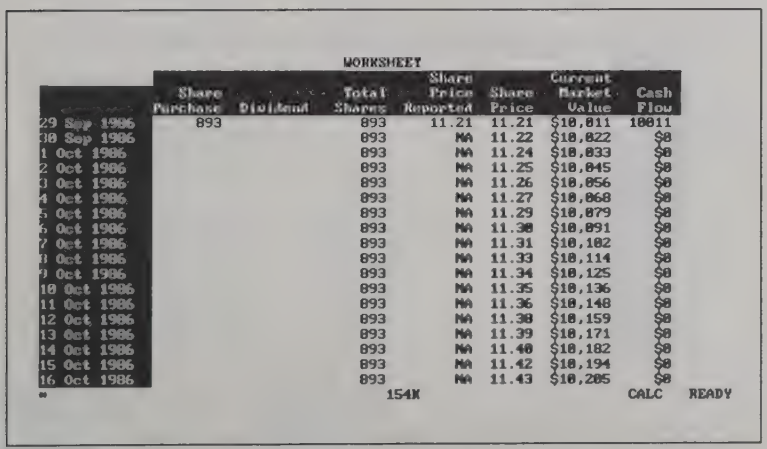


Figure 19.7: 1-2-3 fund spreadsheet imported into worksheet with borders

Unlike the fund spreadsheet in which each date has a single line of information, the checkbook may have several lines for a given date. In Chapter 10 you solved this by using custom time periods. Defining the custom time period is easy: use / **D**efine **T**ime period **A** and change the name to Tr. Since you want to group transactions by month, define Trs in terms of months. The question is, How many Trs are in each month?

If you import this kind of data into Javelin, the following procedure will help you avoid having to count the number of transactions in each month and type them into the Custom Period settings sheet that relates the Tr time period to calendar time. We started with the worksheet as in Figure 19.10.

```
Formulas for all variables:

Cash Flow = - Current Market Value @1 Feb 1987; ((Share Purchase;0) - (Dividend;0)) *
     Share Price
Current Market Value = CUM(Share Purchase; 0) * Share Price
Dividend = < Daily data: 31 October 1986 - 30 January 1987 >
IRR = IRR(BEGIN(29 Sep 1986, Cash Flow), 0.2 / 365) * 365
Share Price = INTERP(Share Price Reported); PREVIOUS(Share Price)
Share Price Reported = < Daily data: 29 September 1986 - 30 January 1987 >
Share Purchase = < Daily data: 29 September 1986 - 29 September 1986 >
```

Figure 19.8: Formulas for FUND Model

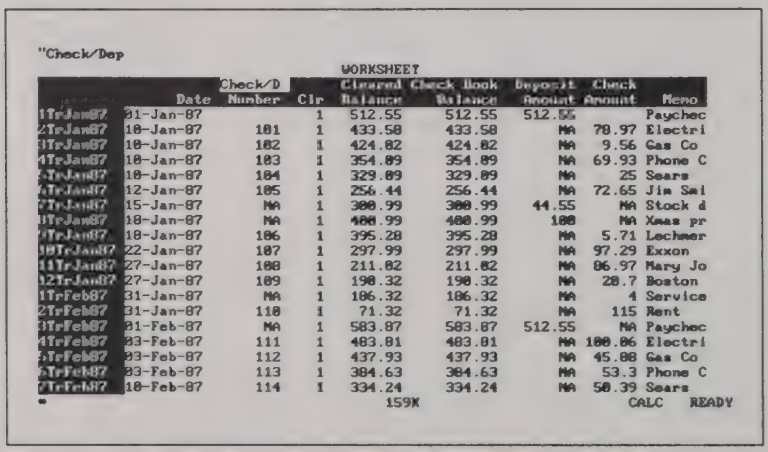
```
                                    WORKSHEET
                    IRR of Cash Flows:        32.82%

                  Share                      Share Price   Share   Current Market   Cash
                  Purchase      Dividend     Reported      Price   Value            Flow
29 Sep 86              893                        11.21     11.21   $10,011      $10,011
30 Sep 86                                         NA        11.22   $10,022          $0
1 Oct 86                                          NA        11.24   $10,033          $0
2 Oct 86                                          NA        11.25   $10,045          $0
  ...           ...          ...          ...     ...       ...     ...
30 Oct 86                                         NA        11.61   $10,365          $0
31 Oct 86                  10.966           11.62     11.62   $10,377       ($127)
1 Nov 86                        0              NA     11.61   $10,370          $0
  ...           ...          ...          ...     ...       ...     ...
29 Jan 87                       0              NA     12.82   $10,735          $0
30 Jan 87                   4.866           12.86     12.86   $10,778       ($49)
31 Jan 87                                         12.86   $10,778          $0
1 Feb 87                                          12.86   $10,778 ($10,778)
FUND2.MDL                                159K                                 READY
```

Figure 19.9: Worksheet summary of FUND model

Date	Check/Dep Number	Clr	Cleared Balance	Check Book Balance	Deposit Amount	Check Amount	Memo
01-Jan-87		1	$512.55	$512.55	$512.55		Paycheck
10-Jan-87	101	1	$433.58	$433.58		$78.97	Electric
10-Jan-87	102	1	$424.02	$424.02		$9.56	Gas Co
10-Jan-87	103	1	$354.09	$354.09		$69.93	Phone Co
10-Jan-87	104	1	$329.09	$329.09		$25.00	Sears
12-Jan-87	105	1	$256.44	$256.44		$72.65	Jim Smith
15-Jan-87		1	$300.99	$300.99	$44.55		Stock div
18-Jan-87		1	$400.99	$400.99	$100.00		Xmas pres
18-Jan-87	106	1	$395.28	$395.28		$5.71	Lechmere
22-Jan-87	107	1	$297.99	$297.99		$97.29	Exxon
27-Jan-87	108	1	$211.02	$211.02		$86.97	Mary Jone
27-Jan-87	109	1	$190.32	$190.32		$20.70	Boston Gl
31-Jan-87		1	$186.32	$186.32		$4.00	Service C
31-Jan-87	110	1	$71.32	$71.32		$115.00	Rent
01-Feb-87		1	$583.87	$583.87	$512.55		Paycheck
03-Feb-87	111	1	$483.81	$483.81		$100.06	Electric
03-Feb-87	112	1	$437.93	$437.93		$45.88	Gas Co
03-Feb-87	113	1	$384.63	$384.63		$53.30	Phone Co
10-Feb-87	114	1	$334.24	$334.24		$50.39	Sears
11-Feb-87	115	1	$294.32	$294.32		$39.92	Bargain B
15-Feb-87		1	$338.87	$338.87	$44.55		Stock div
18-Feb-87	116	1	$266.47	$266.47		$72.40	Lechmere
22-Feb-87	117	1	$203.05	$203.05		$63.42	Exxon
26-Feb-87	118	1	$178.66	$178.66		$24.39	Safeway
28-Feb-87		1	$174.66	$174.66		$4.00	Service C
28-Feb-87	119	1	$59.66	$59.66		$115.00	Rent
01-Mar-87		1	$572.21	$572.21	$512.55		Paycheck
03-Mar-87	120	1	$563.95	$563.95		$8.26	Electric
10-Mar-87	121	1	$486.59	$486.59		$77.36	Gas Co
10-Mar-87	122	1	$437.15	$437.15		$49.44	Phone Co
10-Mar-87	123	1	$416.64	$416.64		$20.51	Sears
15-Mar-87		1	$461.19	$461.19	$44.55		Stock div
15-Mar-87	124	0	$461.19	$407.52		$53.67	Dr. Grimb
18-Mar-87	125	0	$461.19	$314.79		$92.73	Lechmere
22-Mar-87	126	1	$447.05	$300.65		$14.14	Exxon
26-Mar-87	127	0	$447.05	$279.36		$21.29	Safeway
26-Mar-87	128	1	$441.41	$273.72		$5.64	Nan's Not
29-Mar-87	129	0	$441.41	$255.63		$18.09	Kitchen W
31-Mar-87		1	$437.41	$251.63		$4.00	Service C
31-Mar-87		0	$437.41	$291.63	$40.00		Jim Smith
31-Mar-87	130	0	$437.41	$176.63		$115.00	Rent

Table 19.2: Checkbook spreadsheet in 1-2-3

- Using / **D**efine **T**ime period **A**, define custom time period A to have the name you wish to use, in this case, Tr. Also define the time period in terms of the appropriate calendar unit, in this case, months.
- Press Esc to see the Custom Period settings sheet for relating Tr to months. Move to the first month for which you have data (January 1987 in this instance).
- Enter a large number for the number of custom time periods in this month. We used 1000, but any number larger than the number of lines you imported will do.
- Press Esc to exit from this table.
- Use / *Worksheet **B**order **D**ates to fill the left border with dates from 1 Tr to the maximum number you specified. In our example, the dates go from 1 Tr Jan 87 to 1000 Tr Jan 87 (see Figure 19.11).
- Move the cursor down the left border until the last transaction for the first month. We stopped at the end of January (14 Tr Jan 87). This tells you how many custom periods are actually in the calendar period. Remember this number (14 here).
- Use / **D**efine **T**ime period **Tr** and hit Esc to see the Custom Period settings sheet that relates Tr to months. Enter the number you just remembered for this time period and enter your large number for the next time period. Hit Esc to exit from the settings sheet. We entered 14 for January and 1000 for February.



"Check/Dep

WORKSHEET

	Date	Check/D Number	Clr	Cleared Balance	Check Book Balance	Deposit Amount	Check Amount	Memo
1TrJan87	01-Jan-87		1	512.55	512.55	512.55		Paychec
2TrJan87	10-Jan-87	101	1	433.58	433.58	NA	78.97	Electri
3TrJan87	10-Jan-87	102	1	424.02	424.02	NA	9.56	Gas Co
4TrJan87	10-Jan-87	103	1	354.09	354.09	NA	69.93	Phone C
5TrJan87	10-Jan-87	104	1	329.09	329.09	NA	25	Sears
6TrJan87	12-Jan-87	105	1	256.44	256.44	NA	72.65	Jim Sai
7TrJan87	15-Jan-87	NA	1	300.99	300.99	44.55	NA	Stock d
8TrJan87	18-Jan-87	NA	1	400.99	400.99	100	NA	Xmas pr
9TrJan87	18-Jan-87	106	1	395.28	395.28	NA	5.71	Lechner
10TrJan87	22-Jan-87	107	1	297.99	297.99	NA	97.29	Exxon
11TrJan87	27-Jan-87	108	1	211.02	211.02	NA	86.97	Mary Jo
12TrJan87	27-Jan-87	109	1	198.32	198.32	NA	28.7	Boston
1TrFeb87	31-Jan-87	NA	1	186.32	186.32	NA	4	Service
2TrFeb87	31-Jan-87	110	1	71.32	71.32	NA	115	Rent
3TrFeb87	01-Feb-87	NA	1	583.87	583.87	512.55	NA	Paychec
4TrFeb87	03-Feb-87	111	1	483.81	483.81	NA	100.06	Electri
5TrFeb87	03-Feb-87	112	1	437.93	437.93	NA	45.88	Gas Co
6TrFeb87	03-Feb-87	113	1	384.63	384.63	NA	53.3	Phone C
7TrFeb87	10-Feb-87	114	1	334.24	334.24	NA	50.39	Sears

159K

CALC READY

Figure 19.10: Checkbook worksheet after adjusting borders

- Repeat the preceding two steps for each calendar time period for which you have custom period data. If you have many such periods, you may find the macro listed in Figure 19.12 useful. If you type this macro in and call it C (for Custom Time), all you have to do is move the cursor down to the last transaction in your calendar time period and hit Alt-C. The macro will remember the number and enter it into the custom time period to produce the calendar time table.

WORKSHEET

		Date	Check/D Number	Clr	Cleared Check Book Balance	Check Book Balance	Deposit Amount	Check Amount	Memo
1	Tr Jan 87	01-Jan-87		1	512.55	512.55	512.55		Paychec
2	Tr Jan 87	10-Jan-87	101	1	433.58	433.58	NA	78.97	Electri
3	Tr Jan 87	10-Jan-87	102	1	424.02	424.02	NA	9.56	Gas Co
4	Tr Jan 87	10-Jan-87	103	1	354.09	354.09	NA	69.93	Phone C
5	Tr Jan 87	10-Jan-87	104	1	329.09	329.09	NA	25	Sears
6	Tr Jan 87	12-Jan-87	105	1	256.44	256.44	NA	72.65	Jim Smi
7	Tr Jan 87	15-Jan-87	NA	1	300.99	300.99	44.55	NA	Stock d
8	Tr Jan 87	18-Jan-87	NA	1	400.99	400.99	100	NA	Xmas pr
9	Tr Jan 87	18-Jan-87	106	1	395.28	395.28	NA	5.71	Lechmer
10	Tr Jan 87	22-Jan-87	107	1	297.99	297.99	NA	97.29	Exxon
11	Tr Jan 87	27-Jan-87	108	1	211.02	211.02	NA	86.97	Mary Jo
12	Tr Jan 87	27-Jan-87	109	1	198.32	198.32	NA	28.7	Boston
13	Tr Jan 87	31-Jan-87	NA	1	186.32	186.32	NA	4	Service
14	Tr Jan 87	31-Jan-87	110	1	71.32	71.32	NA	115	Rent
15	Tr Jan 87	01-Feb-87	NA	1	583.87	583.87	512.55	NA	Paychec
16	Tr Jan 87	03-Feb-87	111	1	483.81	483.81	NA	100.06	Electri
17	Tr Jan 87	03-Feb-87	112	1	437.93	437.93	NA	45.88	Gas Co
18	Tr Jan 87	03-Feb-87	113	1	384.63	384.63	NA	53.3	Phone C
19	Tr Jan 87	10-Feb-87	114	1	334.24	334.24	NA	50.39	Sears

ck1.MDL∞ 159K CALC READY

Figure 19.11: Checkbook with Tr period dates in left border

```
Macro c:

@REM  Macro to help importing date series into custom time periods.
@REM  How many custom periods should there be in each month?
@REM  Javelin will number them starting at the beginning of the month.
@REM  If you put the cursor on the last number for the month, this
@REM  macro will copy that number into the Custom Time Period settings sheet.


@F2                             @REM Edit the custom time period (it's NN Month Year).
@CTRLRIGHT @CTRLK               @REM Keep the first word, delete the rest of the line
@HOME @CTRLK                    @REM and cut the number into the kill buffer.
@ESC                            @REM Forget all that editing stuff.
@SLASH D T @SPELL @ENTER        @REM Define the first time period.
@ESC                            @REM Just go to the setting sheet.
@UP @CTRLY @ENTER               @REM Get the number we cut; it's the number of periods
                                @REM   for this month.
1000 @ENTER                     @REM Having moved down to the next month, enter a large
                                @REM   number.
@ESC                            @REM And we're done.
```

Figure 19.12: Macro for grouping custom time periods into calendar time

Figure 19.13 shows the completed checkbook model in Javelin. We have taken advantage of Javelin PLUS's ability to store dates in a variable and have translated the 1-2-3 dates into Javelin dates using the macro described earlier. The formulas in this model are listed in Figure 19.14.

■ *Importing DIF and SYLK Files*

Importing these types of files works similarly to importing ASCII Delimited files. Javelin places the imported information on the worksheet starting at your cursor position. Only labels and numbers are imported, without formulas.

Exporting Data

■ Javelin also exports data from the Worksheet view, using the / *Worksheet **X**port commands. Javelin asks for the range of the worksheet you'd like to export and then writes the data in a new file of the indicated format. It writes out the labels and the values in the range and does not export formulas.

WORKSHEET

	Date	Trans Number	Clr	Cleared Check Book Balance	Check Book Balance	Deposit Amount	Check Amount	Memo
1TrJan87	1 Jan 87		1	512.55	512.55	512.55		Paychec
2TrJan87	10 Jan 87	101	1	433.58	433.58		78.97	Electri
3TrJan87	10 Jan 87	102	1	424.02	424.02		9.56	Gas Co
4TrJan87	10 Jan 87	103	1	354.09	354.09		69.93	Phone C
5TrJan87	10 Jan 87	104	1	329.09	329.09		25.00	Sears
6TrJan87	12 Jan 87	105	1	256.44	256.44		72.65	Jim Sai
7TrJan87	15 Jan 87		1	300.99	300.99	44.55		Stock d
8TrJan87	18 Jan 87		1	400.99	400.99	100.00		Xmas pr
9TrJan87	18 Jan 87	106	1	395.28	395.28		5.71	Lechmer
10TrJan87	22 Jan 87	107	1	297.99	297.99		97.29	Exxon
11TrJan87	27 Jan 87	108	1	211.02	211.02		86.97	Mary Jo
12TrJan87	27 Jan 87	109	1	190.32	190.32		20.70	Boston
1TrFeb87	31 Jan 87		1	186.32	186.32		4.00	Service
2TrFeb87	31 Jan 87	110	1	71.32	71.32		115.00	Rent
3TrFeb87	1 Feb 87		1	583.87	583.87	512.55		Paychec
4TrFeb87	3 Feb 87	111	1	483.81	483.81		100.06	Electri
5TrFeb87	3 Feb 87	112	1	437.93	437.93		45.88	Gas Co
6TrFeb87	3 Feb 87	113	1	384.63	384.63		53.30	Phone C
7TrFeb87	10 Feb 87	114	1	334.24	334.24		50.39	Sears

CK1.MDL* 115K READY

Figure 19.13: Finished checkbook model imported from 1-2-3

Javelin's ability to export data also has improved with every release. In fact, Javelin 1.0 had no export facility other than formatting your worksheet and printing it to a file. Here are the selections available on the / *Worksheet **X**port file menu (new to version 1.1) and the version of Javelin in which they first appeared:

- Javelin model: (Javelin 1.1.) Creates a new Javelin model that contains this worksheet and all variables displayed on it. The variables are saved as input variables (not formulas) with their current values for the entire time period for which they have data, not just the time period displayed on the worksheet. (See Chapter 18 for more information.)
- DIF: (Javelin 1.1.) Exports in DIF format with the file extension DIF.
- WKS: (Javelin 1.1.) Exports as a 1-2-3 version 1 spreadsheet file with the file extension WKS. These are readable by all versions of 1-2-3 and Symphony.
- ASCII: (Javelin PLUS.) Exports as an ASCII Delimited file with double quotes around literals and file extension PRN.
- SYLK: (Javelin PLUS.) Exports in Microsoft SYLK format with file extension SYL.

If the worksheet range contains special values, Javelin saves them as labels. #NA is saved as the label NA, #RANGE is saved as a blank, and all other special values are saved as the label ERROR (or as the special value @ERR in 1-2-3).

There is one major difference between exporting to a Javelin model and all other forms of exporting: exporting to a Javelin model is the *only* format

```
Formulas for all variables:

Check Amount = < Tr period data: 2 Tr January 1987 - 15 Tr March 1987 >
Check Book Balance = PREVIOUS(Check Book Balance; 0)+ Deposit Amount -
     Check Amount
Cleared Balance = PREVIOUS(Cleared Balance; 0) + IF(Clr, Deposit Amount -
     Check Amount, 0)
Clr = < Tr period data: 1 Tr January 1987 - 15 Tr March 1987 >
Date = < Tr period data: 1 Tr January 1987 - 15 Tr March 1987 >
Deposit Amount = < Tr period data: 1 Tr January 1987 - 14 Tr March 1987 >
Trans Number = < Tr period data: 2 Tr January 1987 - 15 Tr March 1987 >
```

Figure 19.14: Formulas for finished checkbook model

that exports more information than is on the worksheet itself. That is, if you have monthly data from January 1985 to December 1987 but your worksheet displays only the yearly totals, / *Worksheet **X**port **J**avelin model contains all the monthly data. / *Worksheet **X**port **W**KS creates a 1-2-3 spreadsheet with only the yearly data that appear on the worksheet, as do the other export commands.

■ *Printing to a File*

To export data from a Javelin model to an ASCII Text file, you use a different command entirely. Rather than using / *Worksheet **X**port, you print the worksheet to a file instead of to the printer. This means that the characters that normally would be printed out by the command / **P**rint **C**urrent view are stored in a file instead. Using this feature and the flexibility you have in presenting data on a worksheet, you should be able to export data to almost any program. You may also print to a file from any other view.

To print to a file, use / **D**efine **P**rinter to change the Printer type setting. The last two choices are:

- New file: Creates a new file each time you print. After you give the / **P**rint command, Javelin prompts you for a file name. If you do not enter a file extension, Javelin uses the extension PRN.
- Append to file: Adds the text to the end of an existing file.

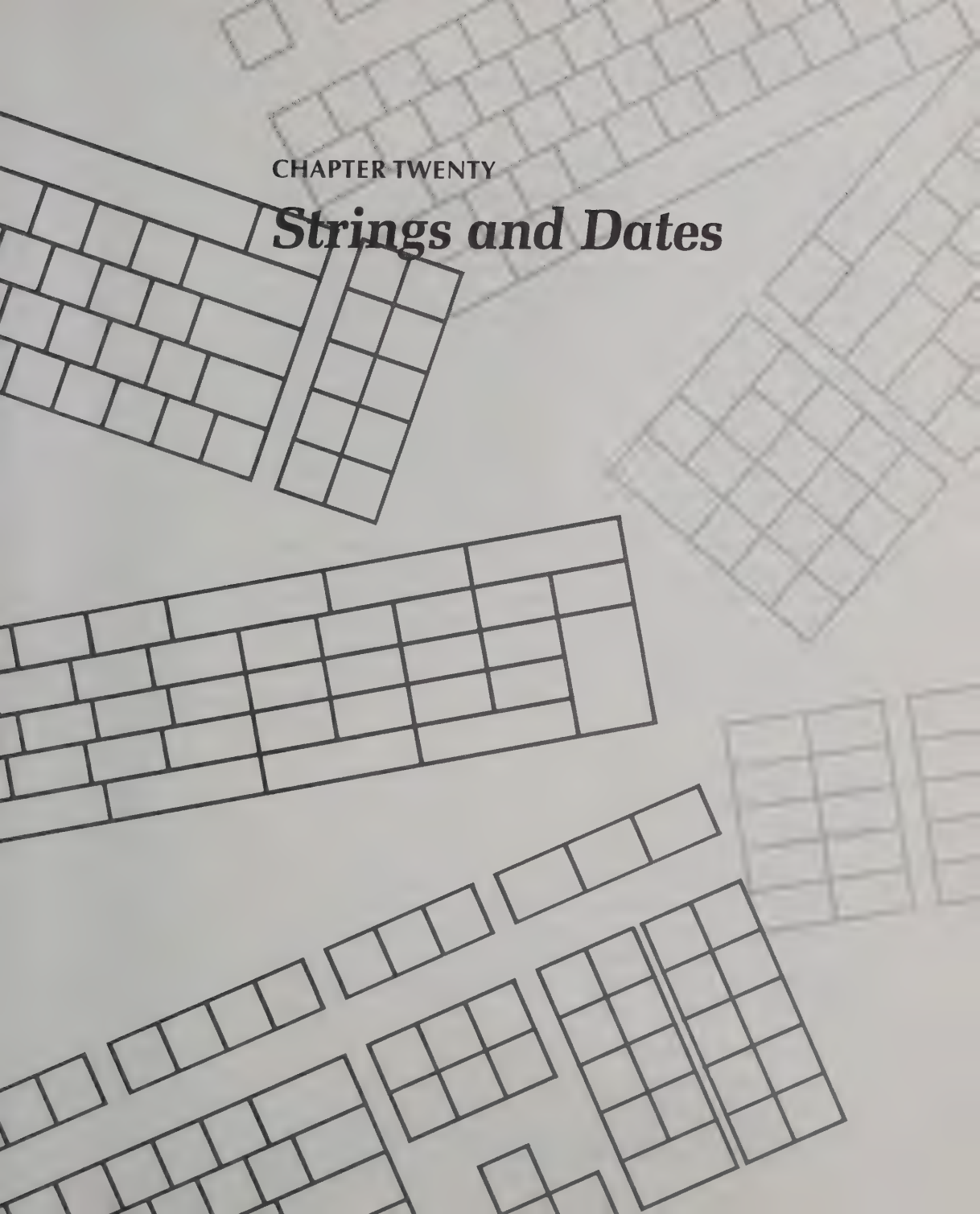
You also want to print to a file when creating documentation for your models. In that case, the Formulas and Notes views can be helpful. In fact, all the formula listings in this book were printed to files directly from Javelin.

Conclusion

■ Javelin's import and export facilities allow you to transfer information to and from many programs. So if you need to exchange information with someone using a different program, it should be fairly simple. In fact, if you set up the borders properly in your worksheet and if the sending program always outputs in the same rows and columns, you can automatically consolidate information in Javelin that is created in a number of different programs. Just give the necessary command, / *Worksheet **I**mport file, and recalculate your model. Because variables are defined as they are read into

the worksheet, they enter right into your model's central information base.

Importing 1-2-3 spreadsheets into Javelin may appear to be equally simple, but converting a spreadsheet to Javelin is more involved, since Javelin imports the *results* of 1-2-3 formulas, not the formulas themselves. In addition, since 1-2-3 and Javelin deal with dates in very different ways, be sure to convert your 1-2-3 dates to text dates before importing them. Finally, as you convert 1-2-3 formulas into Javelin formulas, pause to rethink the logic of your model to take advantage of Javelin's functions and other features.



CHAPTER TWENTY

Strings and Dates

In many models, you may find that you need to keep track of information other than numbers. For instance, in the checkbook model introduced in Chapter 10, you entered a Check Number, Check Date, Check Amount, and Check Who (to whom it is made out) for each check. Only Check Number and Check Amount were actual Javelin variables. Check Date and Check Who were stored only as labels on a worksheet, because Check Date is a date and Check Who is text information. As worksheet labels, there was no way to manipulate these values—for example, no way to look at a daily series of expenditures, or checks by the payees.

In Javelin PLUS, variables may contain numeric, text, or date values, so Check Date and Check Who can become variables like Check Amount and Check Number. In fact, anywhere you can type in a value for a variable, you can enter text or a date instead of a number. Javelin PLUS also has a complete set of operators and functions that allow you to manipulate text and dates, and convert one type of data to another.

In this chapter, we'll use a mailing list to illustrate the use of Javelin PLUS's text and date values, and their operators and functions. In Chapter 21, "Javelin Databases," you'll learn to use these text and date functions to implement a database. If you don't have Javelin PLUS, you won't be able to accompany the exercises in these chapters—the addition of text and dates is one of the main reasons that upgrading to Javelin PLUS is worthwhile.

Text Values: Strings

■ Javelin PLUS stores its text information (also called strings) surrounded by label prefixes—apostrophes, double quotes, carets, or backslashes. Label prefixes were used in previous versions of Javelin to tell the Worksheet view how to display a label, but they weren't needed at the end of the string. The label prefix still determines how the string will be displayed in cells in the Worksheet view:

- Apostrophes: left justified;
- Double quotes: right justified;
- Carets: centered;
- Backslashes: repeated to fill the cell.

If you do not type a label prefix, Javelin uses apostrophes. If you type one label prefix at the beginning of the string and another at the end, Javelin uses the one at the beginning to position it in the cell in the Worksheet view.

You can enter strings for the values of variables just as you have entered numeric values, in the Table and Worksheet views. You cannot view or edit strings in the Chart, Quick-graph, and Graph views, since these views work with graphic depictions of numeric values. Strings show up as zeros in these views.

First, enter some string data. Starting with a blank model, begin the mailing list model, MAILLIST. In this model, you will store First Name, Last Name, Address, City, State, Zip, and Phone for each person on the list. You may recall from Chapter 10 that Javelin PLUS has an added time period, called Record. Unlike custom time periods A and B, the Record period is never related to time. Record was added to Javelin so models that already use their A and B time periods can add nontime-related string and date information without conflicting with their existing time periods, and also to emphasize that a list of records is not a time series. You can use strings and dates in daily, weekly, and other normal time series as well—for example, to store the name of the manager in charge for each day.

To begin, use the command / **D**efine **S**ettings to display the Model settings sheet, make Record the default time period, and set the model's name as MAILLIST. Then display a Table view of a variable called First Name. Enter these names:

1 Record: Nick
2 Record: Rocky
3 Record: Susan
4 Record: George
5 Record: Clem
6 Record: John

Your screen should look like Figure 20.1. Note that Javelin has added apostrophes to the beginning and end of each string.

Display a worksheet called List of Names and use / *Worksheet **B**order **D**ates to enter 1 Record through 6 Record in the left border. In the top border of the first column of body cells, enter

First Name

The first names should appear. Although the apostrophes are still attached to your text, they don't show in the Worksheet view—they are used

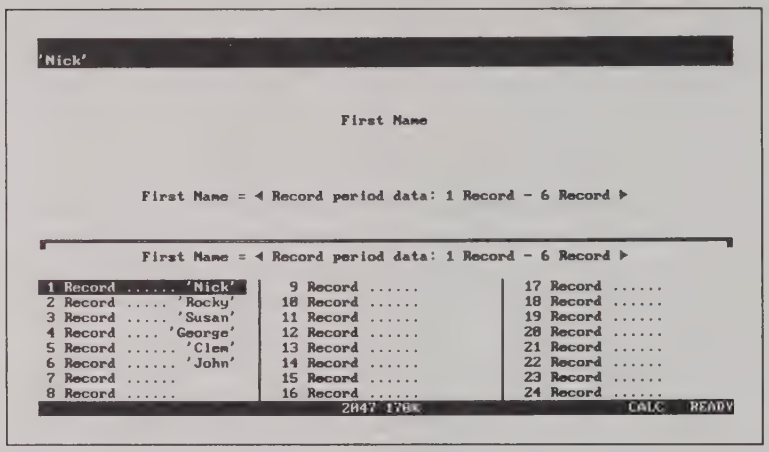


Figure 20.1: Entering text values in the Table view

just to position the text in the cell. When your cursor is on a cell that contains a string, you still see the label prefixes surrounding your text on the edit line.

In the next column, enter

Last Name

and enter these names in the body cells:

1 Record: Danger
2 Record: Rococco
3 Record: Underhill
4 Record: Tyrebiter
5 Record: Ah
6 Record: Firesign

Now you have two input variables with string values, First Name and Last Name. To complete the data for this mailing list, enter the variables and data shown in Figure 20.2, so that you have Address, City, State, Zip, and Phone for each person.

■ *String Operators*

Once you have entered string information into your Javelin model, you can do calculations with those strings. Just as you do with numbers, you can

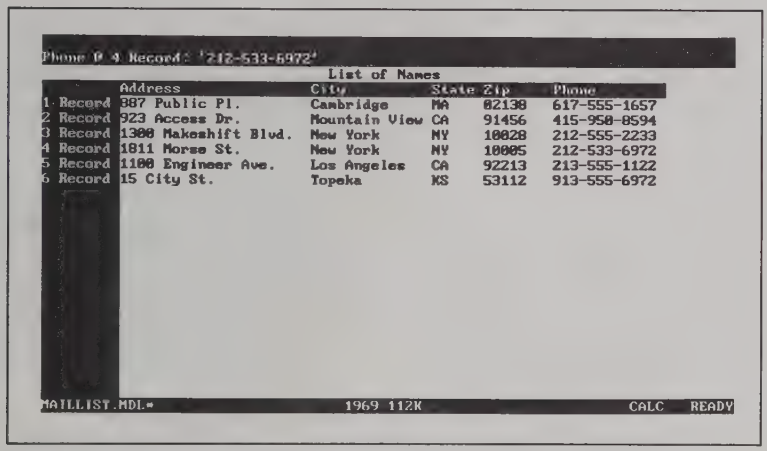

Phone @ 4 Record: "212-533-6972"
List of Names
Address City State Zip Phone
1 Record 387 Public Pl. Cambridge MA 02138 617-555-1657
2 Record 923 Access Dr. Mountain View CA 91456 415-958-8594
3 Record 1300 Makeshift Blvd. New York NY 10028 212-555-2233
4 Record 1811 Morse St. New York NY 10005 212-533-6972
5 Record 1100 Engineer Ave. Los Angeles CA 92213 213-555-1122
6 Record 15 City St. Topeka KS 53112 913-555-6972
MAILLIST.MDL* 1969 112K CALC READY

Figure 20.2: Input variables for mailing list

enter formulas that use *string operators* and string functions. Javelin has seven string operators, which function similarly to their counterparts for numbers. They allow you to "add" one string to the end of another and to test the relationship between two strings.

- Plus (+): *Concatenates* two strings; that is, one string is added onto the end of the other string. Javelin uses the label prefix from the first string of the concatenation.
- Equal (=): Compares two strings to see if they match. Capitalization doesn't count (see the EXACT function later in this chapter). Returns 1 (true) if they match and 0 (false) otherwise.
- Greater than (>): Returns 1 (true) if the first string is greater than the second (comes later in alphabetical order) and 0 (false) otherwise.
- Less than (<): Returns 1 (true) if the first string is less than the second (comes earlier in alphabetical order) and 0 (false) otherwise.
- Greater than or equal (>=): Returns 1 (true) if the first string is greater than or matches the second and 0 (false) otherwise.
- Less than or equal (<=): Returns 1 (true) if the first string is less than or matches the second and 0 (false) otherwise.
- Not equal (<>): Returns 1 (true) if the two strings don't match and 0 (false) otherwise.

The comparison operators do not look at the label prefixes of the two strings when comparing them.

If you add or subtract a string and a number, Javelin treats the string as if it were a zero and does the calculation. For example, Javelin evaluates

```
First Name + 3
```

as equal to 3. If you multiply or divide a number and a string, the result is #BLANK. Tables 20.1 and 20.2 detail what happens if you mix numbers and strings in formulas.

Try some string operators. Define a calculated variable called Name. In the top border of the next column on the worksheet, enter

```
Name = First Name + Last Name
```

When you press F9, the Calc key, you'll see these values appear, but with no space between the two parts of the name. To make this look better, edit the formula to read

```
Name = First Name + " " + Last Name
```

Operator	Two Strings	A Number and a String	Two Dates	A Number and a Date	A String and a Date
+	Concatenates	Number	#NODATE	Date	#NODATE
-	#BLANK	-Number	Number (if same periods)	Date	#NODATE
*	#BLANK	#BLANK	#NODATE	#NODATE	#NODATE
/	#BLANK	#BLANK	#NODATE	#NODATE	#NODATE
^	#ARITH	#ARITH	#NODATE	#NODATE	#NODATE
= < > <= >= <>	Compares	See note	Compares (if same periods)	See note	See note

Note: In comparisons, numbers are greater than dates or strings, and strings are greater than dates.

Table 20.1: What operators and functions do with mixed data in Javelin PLUS

This concatenates a space to First Name and then concatenates Last Name to the result. Now when you recalculate, your screen should look like Figure 20.3. It doesn't matter whether you enclose the space in apostrophes or double quotes.

To create a variable whose value is true for people on your list who live in Massachusetts, for example, enter the following in a top border cell:

```
In Mass = (State = 'MA')
```

You must put the comparison in parentheses so Javelin knows which equal sign is for comparison and which defines the formula. The results are ones

Numeric Function	All Strings	Numbers and Strings	All Dates	Numbers and Dates	Strings and Dates
AVG TAVG RAVG	#ARITH #BLANK	Ignores strings and averages numbers	#NODATE	#NODATE	#NODATE
MIN	First in sort order	First in sort order	First in sort order	First in sort order	First in sort order
TMIN RMIN	First in sort order	First in sort order	First in sort order	Indeter-minate	Indeter-minate
MAX	Last in sort order	Last in sort order	Last in sort order	Last in sort order	Last in sort order
TMAX RMAX	Last in sort order	Last in sort order	Last in sort order	Indeter-minate	Indeter-minate
SORTA	Sort order	Sort order	Sort order	Sort order	Sort order
SORTD	Reverse sort order	Reverse sort order	Reverse sort order	Reverse sort order	Reverse sort order

SUM, TSUM, RSUM, CUM, MTD, QTD, and YTD give the same results as the equivalent formula using plus signs.

Javelin sort order is #BLANK, followed by strings (alphabetical order), followed by dates (earliest to latest), followed by numbers (lowest to highest), followed by special values like #ERROR and #NA.

Table 20.2: What numeric functions do with mixed data in Javelin PLUS

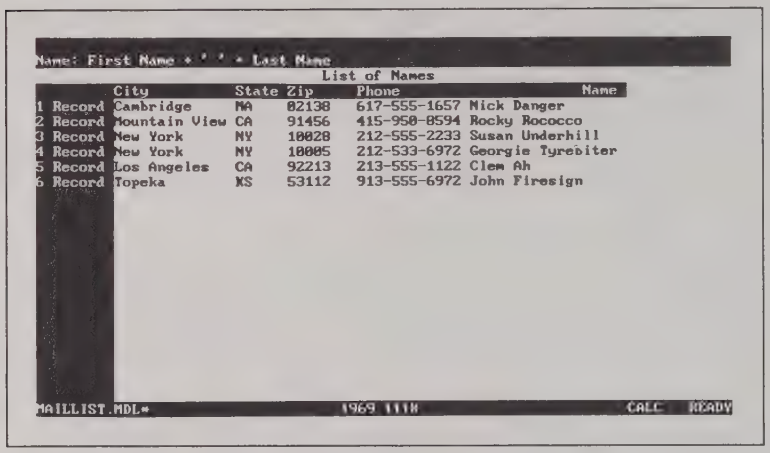

Name: First Name + ' ' + Last Name
List of Names
City State Zip Phone Name
1 Record Cambridge MA 02138 617-555-1657 Nick Danger
2 Record Mountain View CA 91456 415-950-8594 Rocky Rococco
3 Record New York NY 10028 212-555-2233 Susan Underhill
4 Record New York NY 10005 212-533-6972 Georgie Tyrebiter
5 Record Los Angeles CA 92213 213-555-1122 Clem Ah
6 Record Topeka KS 53112 913-555-6972 John Firesign
MAILLIST.MOD*
1969 111K
CALC
READY

Figure 20.3: Concatenating two strings

(true) and zeros (false). Change one state to 'Ma' and see what happens—nothing. Javelin ignores capitalization when comparing strings.

Similarly, you can create a variable that is true for people whose names begin in the second half of the alphabet. Use this formula:

Second Half = (Last Name >= 'N')

By themselves, these variables are not particularly useful. However, combined with some of the functions listed below and the database features of the next chapter, they can be. More likely, you will want to include formulas like these *inside* the functions described below. That way, Javelin can work on strings that meet the conditions you describe.

String Functions

- Several functions accept strings as arguments. They process the text value by changing its length, removing trailing spaces, selecting the first or last portion of a string, or matching one string against another.

■ *Using Part of a String*

To select just part of a string value, use one of these functions:

- LEFT(string, n): Returns the leftmost n characters of the string.
- MID(string, offset, n): Returns the n characters starting at the given offset (position) in the string. The offset is counted from the left, starting with 1.
- RIGHT(string, n): Returns the rightmost n characters of the string.

The resulting strings have the same label prefixes as the input strings. For example, each formula on the left below returns the string on the right:

LEFT('abcdefghij', 4) 'abcd'
MID("abcdefghij", 2, 3) "bcd"
RIGHT(^abcdefghij^, 5) ^fghij^

You can use these functions to pick apart the telephone numbers in your mailing list. Enter these formulas in the top border of the worksheet:

Area Code = LEFT(Phone, 3)
Exchange = MID(Phone, 5, 3)

Your worksheet should look like Figure 20.4.

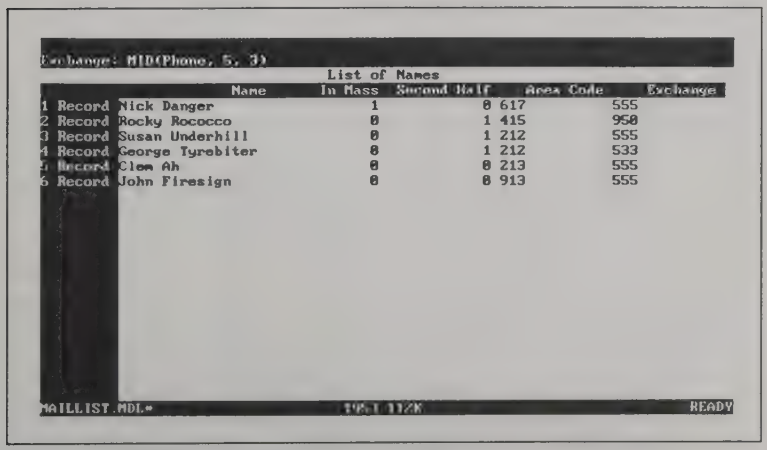

Exchange: MID(Phone, 5, 3)
List of Names
Name
In Mass
Second Half
Area Code
Exchange
1 Record Nick Danger 1 0 617 555
2 Record Rocky Rococco 0 1 415 950
3 Record Susan Underhill 0 1 212 555
4 Record George Tyrebiter 0 1 212 533
5 Record Clem Ah 0 0 213 555
6 Record John Firesign 0 0 913 555
MAILLIST.MDL*
READY

Figure 20.4: LEFT, MID, and RIGHT string functions

■ Comparing, Finding, and Replacing

You may need to take capitalization into account when comparing strings. If so, you can use EXACT.

- EXACT(string1, string2): Returns 1 (true) if the strings match exactly and 0 (false) otherwise. Capitalization counts, as do extra spaces.

Edit the formula In Mass to read

```
In Mass = EXACT(State, 'MA')
```

The state you changed to 'Ma' no longer matches. Change it back to 'MA'.

You can also compare a string to part of another string, to find out if one string contains another. A part of a string is called a *substring*.

- FIND(substring1, string2, n): Searches for substring1 in string2, starting at offset n. It returns the starting position of substring1 in string2, if it finds it. Otherwise, it returns zero.

To search all of string2 for substring1, specify an offset of 1. As with the equal sign, FIND ignores capitalization and label prefixes. To create a variable that records whether or not the abbreviation 'St.' appears in each address, try this formula:

```
Street = FIND('St.', Address, 1)
```

Street is true if FIND found it. If you capitalize the T in one of the addresses, FIND returns the same numbers—Javelin accepts 'ST.' as 'St.'

The REPLACE function lets you substitute one string for part of another:

- REPLACE(string1, offset, n, string2): Returns string1, but with the n characters starting at the offset replaced with string2. If the offset is zero, it returns string1 unchanged.

For example, replace the first dash in each phone number with a slash, to make the numbers look better. In the top border of your worksheet, enter

```
Phone2 = REPLACE(Phone, 4, 1, '/')
```

This replaces the dash at position 4 in each phone number with a slash.

You can use REPLACE and FIND together to replace a substring that is not at a specific offset. For example, to replace the abbreviation 'St.' with Street in the Address, enter

```
Address2 = REPLACE(Address, FIND('St.', Address, 1), 3, 'Street')
```

FIND gives the offset where 'St.' begins. REPLACE takes out the three characters starting at that offset (because 'St.' is three characters long) and substitutes 'Street' at that location. If FIND can't find 'St.' in Address, it returns zero, and if you tell REPLACE to begin replacing starting at character zero, it doesn't replace anything. FIND and REPLACE work well together.

■ *Using Character Codes*

If you need special characters, you may want to convert between strings and their character codes, which are integer numbers between 0 and 255. The characters from 40 to 127 use the industry-standard ASCII code. The characters that appear for the codes from 0 to 31 and from 128 to 255 were defined by IBM when they produced their first PC and are known as the IBM extended graphics character set. Therefore they may print differently on your printer; in fact, many printers use some of these characters to control printer features.

- CHAR(code): Returns the character defined by the given code.
- CODE(string): Returns the code of the first character of the string.

For example, CODE('ABCDE') returns 65, because 65 is the ASCII code for a capital A. CHAR(3) returns a character in the shape of a heart because 3 is the IBM extended graphics character code for a heart. Figure 20.5 illustrates how they appear on the screen. CHAR returns the character as a string, with apostrophes as the label prefix. CODE ignores the label prefix.

If you want to include extended graphics characters in your strings, you can also type them in directly, using the Alt key and the numbers on the numeric keypad. (This works on IBM computers and most compatibles.) Hold down Alt, type the code using the numeric keypad, then release Alt.

Another way to do this is to use F9 to calculate a CODE function while you are typing it in. If you type

```
CODE(251)
```

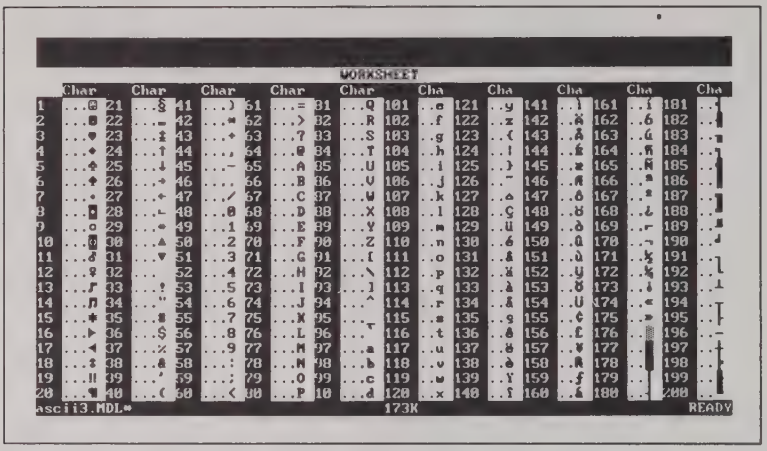


Figure 20.5: Graphics characters returned by CODE function

and press F9 while still on the edit line, Javelin evaluates the CODE function and replaces it with a check mark (for which 251 is the extended character code). You can then continue to type in the string.

■ *Other String Functions*

Three other functions are useful when working with strings:

- REPEAT(string1, n): Returns a string that consists of string1 repeated n times.
- STRLEN(string): Returns the length of the string.
- TRIM(string): Returns a copy of the string without trailing blanks.

For example, if you need a string of 24 spaces to concatenate with another string, rather than typing the 24 spaces you can have Javelin do it:

```
Spaces = REPEAT(' ', 24)
```

The resulting string has the same label prefix as the first argument.

If your First Name values might have unwanted spaces at the ends, you can use this formula to make sure that just one space is between the names:

```
Name = TRIM(First Name) + ' ' + Last Name
```

By using strings, you can include all sorts of non-numeric information in your models, such as category codes, payees of checks, and budget categories.

Date Values

■ Javelin 1.1 manipulates dates in many ways, but has no direct way to store a date in a variable. In Javelin PLUS, you can enter dates as variable values, using the same rules as for entering dates in worksheet border cells.

Javelin PLUS encloses dates in brackets. This tells Javelin that you are typing a date rather than a number or a string. There are many situations in which Javelin wouldn't be able to guess which you mean without the brackets. For example, when entering yearly dates, if you type

```
1988
```

into the body cell of a worksheet or into a Table view, Javelin thinks that you are entering a number (1988.00). You must enclose it in brackets to indicate that it is a date:

```
[1988]
```

Use brackets any time you type a date in the Worksheet view or include a date in a formula.

Add a date variable to the mailing list model. On the List of Names worksheet, in the top border of the next column, enter

```
Renewal Date
```

Enter the following values in the body cells of the same column:

```
1 Record: [12 Jan 88]
2 Record: [31 Mar 88]
3 Record: [14 Feb 88]
4 Record: [29 Jan 88]
5 Record: [3 Mar 88]
6 Record: [29 Feb 88]
```

This creates an input variable with date values. Values for date variables can be entered in the Table and Worksheet views. You can also set up

calculated variables using date operators and date functions. Like strings, dates cannot be handled in the Chart, Quick-graph, and Graph views. Since these views work only with numbers, dates appear as zeros in them.

■ Date Operators

As with strings, Javelin uses as many regular arithmetic operations on dates as it can. Using subtraction you can find the number of days between two dates; using addition you can find a date that is a certain number of days in the future (or the past). You can also use comparison operators to compare two dates. Here are the operators you can use on dates:

- Plus (+): Adds a number to a date, adding that number of days to the date, and returns a date. For weekly, monthly, quarterly, or yearly dates, it adds that many weeks, months, quarters, or years to the date.
- Minus (−): Subtracts one date from another, returning the number of days between the dates, or subtracts a number (of days) from a date, returning a date. For dates in other periods, it returns the number of periods between two dates that you subtract, or subtracts the number of periods from the date.
- Equal (=): Compares two dates to see if they are the same, returning 1 (true) or 0 (false).
- Greater than (>): Compares two dates to see if the first is greater (later) than the second, returning 1 (true) or 0 (false).
- Less than (<): Compares two dates to see if the first is less (earlier) than the second, returning 1 (true) or 0 (false).
- Greater than or equal (>=): Compares two dates to see if the first is greater than or matches the second, returning 1 (true) or 0 (false).
- Less than or equal (<=): Compares two dates to see if the first is less than or matches the second, returning 1 (true) or 0 (false).
- Not equal (<>): Compares two dates to see if they are different, returning 1 (true) or 0 (false).

You cannot multiply or divide dates by numbers or other dates, nor can you add two dates together. Tables 20.1 and 20.2 show what happens if you mix dates, strings, and numbers in formulas.

To find a date that is a certain number of days away from a given date, you can add a number to the date, using the plus sign. For example,

```
[2 Feb 89] + 17
```

results in the date value [19 Feb 89]. You can also subtract a number from a date. For example,

```
[2 Feb 89] - 4
```

results in the date value [29 Jan 89].

To find the number of days from one date to another, subtract the earlier date from the later one, using the minus sign. For example,

```
[15 Apr 88] - [25 Dec 87]
```

results in the numeric value 112.

You can find out which date is later by using the comparison operators. For example, if you want to mail an annual supplement to people still active on a certain date, you could define a variable called Annual Supplement to tell who should receive it. The formula would be

```
Annual Supplement = (Renewal Date < Supplement Date)
```

This formula is true if the Renewal Date is before the Supplement Date.

Date Functions

■ In addition to doing date arithmetic, Javelin PLUS has new functions that accept dates as arguments or return dates. These functions can tell you when a series begins or ends, which week a day falls in, today's date, and other date-related facts.

■ *Today's Date*

One of the most useful functions is today's date. This function allows you to do computations in formulas that include today's date. For instance,

you might want to calculate interest up to today or find out what dates are before or after today.

- TODAY( ): Returns today's date, according to the computer's system clock.

If your PC has no clock/calendar, be sure to set the date and time before running Javelin, using the DOS DATE and TIME commands. (If you're already in Javelin, you can change the system's date by typing / **F**ile **R**un Dos **DATE**.) To find out how many days each person has before the renewal date, use this formula:

Days Left = Renewal Date - TODAY( )

It will be negative if the renewal date has already passed (see Figure 20.6).

Think how handy this feature would be in the FUND model introduced in Chapter 16. You had formulas for IRR and Cash Flow that looked like

IRR = IRR(Cash Flow, 0.2 / 365) * 365
Cash Flow = Current Market Value @ [1 Feb 87]; Share Purchase - Dividend

To change the date for which you wanted the IRR, you had to edit the Cash

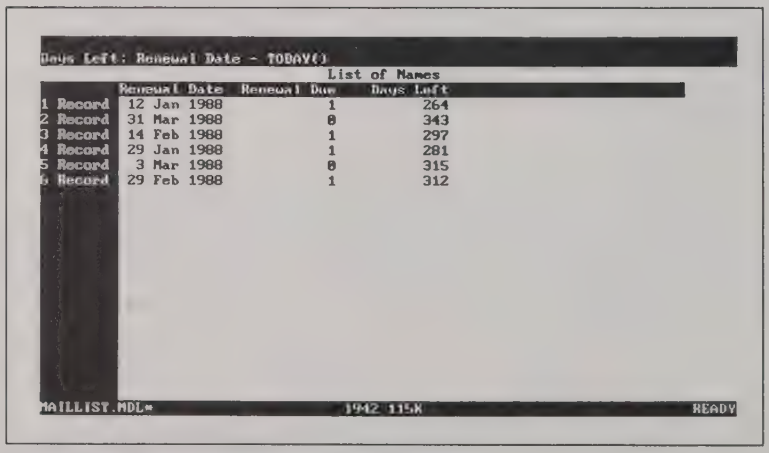

Days Left: Renewal Date - TODAY()
List of Names
Renewal Date
Renewal Due
Days Left
1 Record 12 Jan 1988 1 264
2 Record 31 Mar 1988 0 343
3 Record 14 Feb 1988 1 297
4 Record 29 Jan 1988 1 281
5 Record 3 Mar 1988 0 315
6 Record 29 Feb 1988 1 312
MAILLIST.MDL*
1942 115K
READY

Figure 20.6: Date arithmetic using TODAY function

Flow formula. With the TODAY function, you can get the IRR as of today by changing the Cash Flow formula to be

Cash Flow = Current Market Value @ TODAY( ); Share Purchase – Dividend

■ *Date a Series Begins or Ends*

In previous versions of Javelin, it was difficult to find out when a series started or ended, a fact you sometimes need to know when setting up a formula. There are four date functions for this purpose, all of which return a single date.

- BEGINDATE(series): First date for which the series is defined.
- ENDDATE(series): Last date for which the series is defined.
- FIRST(series): First date for which the series has a nonzero value.
- LAST(series): Last date for which the series has a nonzero value.

The names of these functions are not particularly descriptive, but they are useful. Think of BEGINDATE and ENDDATE as counterparts to the BEGIN and END functions. While BEGIN and END force a series to begin or end at a particular date, BEGINDATE and ENDDATE tell you what that date is. In other words, they do not look at what is *in* the series, they look at *when* the series has some value. FIRST and LAST, on the other hand, look *down* the series, telling you at *what date* the series is not zero.

For example, suppose you want to see if any of your renewal dates are in 1987. The first thing to realize is that the time dimension of your data is [1 Record] to [6 Record]. So FIRST in this case will refer to Record time periods. With this in mind, you can create a formula that uses a date comparison to set Renewals 87 to 1 (true) if there are any renewals in 1987 and 0 (false) otherwise:

Renewals 87 = (FIRST(Renewal Date < [1 Jan 88]) <> #NODATE)

The series Renewal Date < [1 Jan 88] is 1 at each Record period where the renewal date is before 1988 and 0 elsewhere. So the FIRST of this series is the date (like [5 Record]) of the first Renewal Date less than 1988. If no date in [1 Record] through [6 Record] is less than 1988, the answer is #NODATE. Since all your dates are in 1988, Renewals 87 is false.

FIRST and LAST are more useful if your list is sorted into order (something you'll learn to do in the next chapter). Assuming your list is in increasing order by Renewal Date, the following formula tells you how many renewals you have in February 1988:

```
February 88 Renewals = LAST(Renewal Date <= [28 Feb 88]) -
                       FIRST(Renewal Date >= [1 Feb 88])
```

■ First Period in a Larger Period

You may want to find out on what day a particular period starts. For example, for calculations or a report, you may want to know what the first day of [38 W 1988] is, what month that week is in, or what the first day of [3 Q 1988] is. The following functions allow you to answer these questions; they all take a date as their argument and return a date.

- DAY(date value): Returns the first day in the date value.
- WEEK(date value): Returns the first week in the date value.
- MONTH(date value): Returns the first month in the date value.
- QUARTER(date value): Returns the first quarter in the date value.
- YEAR(date value): Returns the year of the date value.
- ACON(date value): Returns the first A custom period in the date value. The name of this function changes if you change the name of the A custom period.
- BCON(date value): Returns the first B custom period in the date value. The name of this function changes if you change the name of the B custom period.

Table 20.3 shows how these functions work.

You can use the DAY function to improve the readability of worksheets that use weekly data. If you need to report data weekly, the way to do so in Javelin is to have borders specifying [1 W 88], [2 W 88], etc. While accurate, this is not the way people are used to looking at dates. Using the DAY function, you can create a column in your worksheet called Week Beginning which displays [27 Dec 87], [3 Jan 88], [10 Jan 88], and others. You can then

print your worksheet without borders so that the Week Beginning column serves as the left border of the report.

Here's how to do this. The following formula creates a variable whose value in a given week is the date of that week:

Week Being Calculated = [1 W 88] + SINCE([1 W 88])

In this way any formula can use the date for which it is being calculated to affect its result. You could write a similar formula for month and day. Once you have Week Being Calculated, you can find the first day of that week using the DAY function:

Week Beginning = DAY(Week Being Calculated)

You can express the same thing using only one variable:

Week Beginning = DAY([1 W 88] + SINCE([1 W 88]))

Date	DAY(Date)	WEEK(Date)	MONTH(Date)	QUARTER(Date)	YEAR(Date)
12 Jan 88	12 Jan 88	3 W 1988	Jan 1988	1 Q 1988	1988
14 Feb 88	14 Feb 88	8 W 1988	Feb 1988	1 Q 1988	1988
29 Feb 88	29 Feb 88	10 W 1988	Feb 1988	1 Q 1988	1988
31 Mar 88	31 Mar 88	14 W 1988	Mar 1988	1 Q 1988	1988
2 Apr 88	2 Apr 88	14 W 1988	Apr 1988	2 Q 1988	1988
2 W 1988	3 Jan 88	2 W 1988	Jan 1988	1 Q 1988	1988
12 W 1988	13 Mar 88	12 W 1988	Mar 1988	1 Q 1988	1988
Feb 1988	1 Feb 88	6 W 1988	Feb 1988	1 Q 1988	1988
Apr 1988	1 Apr 88	14 W 1988	Apr 1988	2 Q 1988	1988
1 Q 1988	1 Jan 88	1 W 1988	Jan 1988	1 Q 1988	1988
2 Q 1988	1 Apr 88	14 W 1988	Apr 1988	2 Q 1988	1988
1988	1 Jan 88	1 W 1988	Jan 1988	1 Q 1988	1988

Table 20.3: DAY, WEEK, MONTH, QUARTER, and YEAR functions

■ Converting Among Dates, Text, and Numbers

Now that you can manipulate text strings in Javelin, an occasion will certainly come when you have a number embedded in a string or a number that should become part of a string. Similarly, with dates you may find that by doing string operations, you have created something that looks like a date and you need Javelin to process the date you have created. You may also want, for example, to extract the month name from the middle of a date. Javelin has a full set of conversion functions to make this easy.

- DATE(string): Converts a string into a date. This works only if the string contains characters that Javelin can interpret as a date. The rules for what Javelin can interpret are the same as those for what you can type in as a date, described in Chapter 8.
- NUM(string): Converts a string into a numeric or date value. This works only if the string contains characters that Javelin can interpret as a number. The rules for what Javelin can interpret are the same as those for what you can type in as a number, described in Chapter 8: the digits 0–9, plus, minus, and decimal point.
- STRING(value): Converts a nonstring value (a number or a date) into a string. The value is converted using the model default date format.

Notice that there are no functions for converting dates to numbers and back again. Since you can do arithmetic with dates there is no need for such conversions. For example, if you have a variable that contains the date [1988] and you need to use the number 1988 in a calculation, you can use the expression 1950 + BETWEEN([1950], [1988]). Similarly, if you need to convert 1988 to a date, use [1950] + (1988–1950). In general, you do many number-to-date and date-to-number conversions. Just write your formulas as they come naturally and Javelin almost always will give you the answer you intend.

You can also create strings that give date information, for example,

```
Day of Week = MID("MonTueWedThuFriSatSun", MOD(SINCE([4
              Jan 1904]), 7) * 3 + 1, 3)
```

This formula relies on the fact that January 4, 1904, was a Monday. Dividing the number of days since then by 7 gives a remainder of 0 for a Monday, up to 6 for a Sunday. The MOD function gives that remainder. You then use that number to pick which three letters to take out of the names of the days of the week.

Another reason to convert a number to a string is to concatenate it with other strings. For example, if you have a numeric variable Credit Limit as part of your mailing list, you might want the following to appear on a worksheet:

```
Name + ' has a credit limit of $' + STRING(Credit Limit) + '.'
```

This formula would result in something like

Fred Smith has a credit limit of $2500.00.

The STRING function can be used to convert dates and numbers into strings, usually to concatenate them with other strings.

```
Message = 'Your renewal date is ' + STRING(Renewal Date) + '.'
```

would calculate as

Your renewal date is 29 Mar 1989.

■ *Determining Types of Values*

Because Javelin now can handle three types of data, it has a TYPE function that lets you find out what type of data a value is.

- TYPE(value): Returns a numeric value: 1 if the argument is numeric, 2 if it is a string, 3 if it is a date value, and 4 if the value is a special value.

Table 20.4 shows the types of some values.

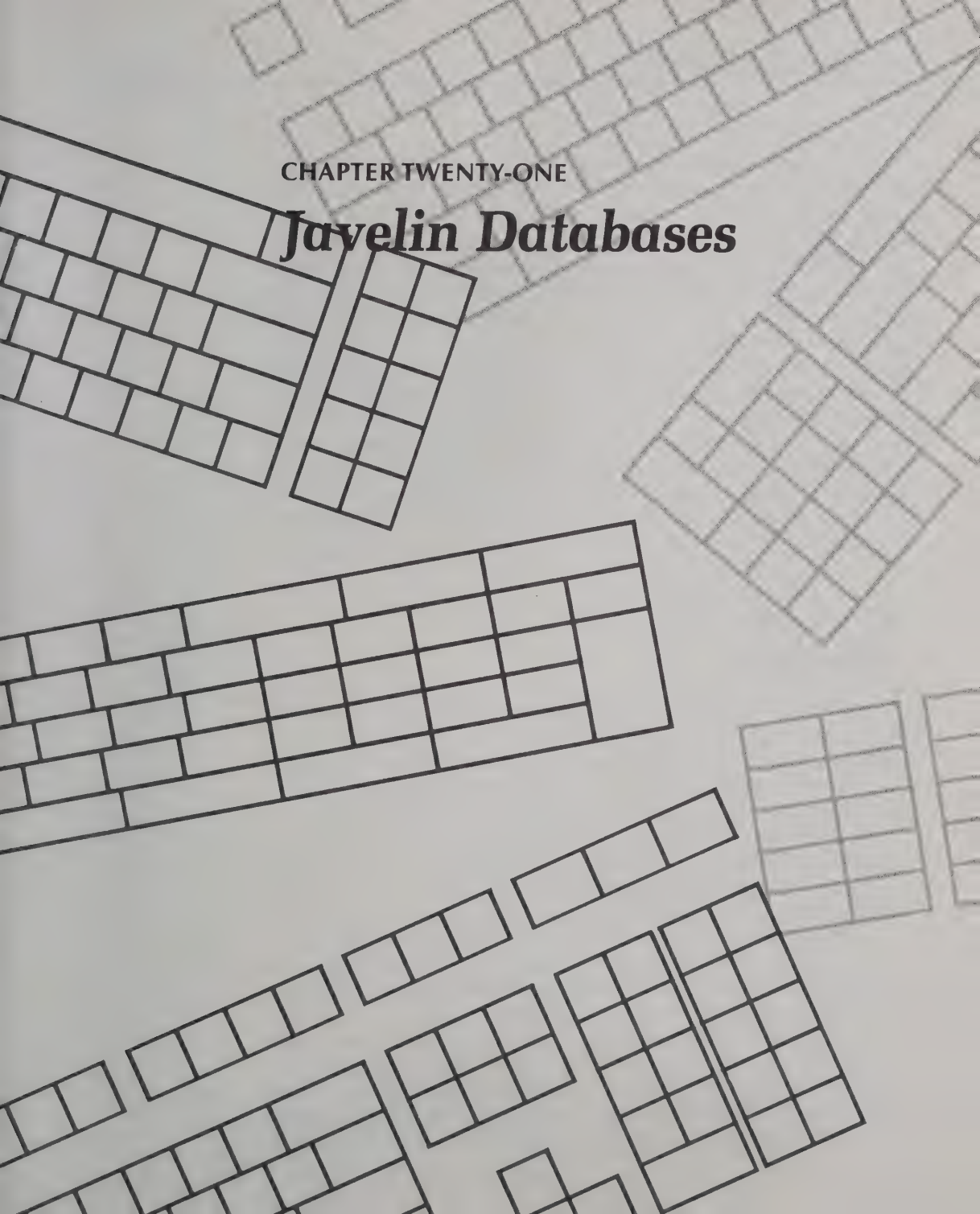
Variable Name	Value at 1 Record	Type
First Name	Nick	2
Last Name	Danger	2
Zip	34234	2
Phone	617-555-1657	2
Renewal Date	12 Jan 1988	3
Credit Limit	12,000	1

Table 20.4: The TYPE function

Conclusion

■ The addition of strings and dates to Javelin opens interesting possibilities in modeling. In this chapter we have just looked at the basic tools provided for using strings and dates. Along the way, we have created a few useful formulas.

Javelin is powerful enough to manage entire databases, which is the subject of the next chapter. Javelin databases are built on the foundation of the string and date functions introduced here. Even if you don't plan to do a full-blown database in Javelin, we strongly recommend that you read the next chapter. The techniques introduced there will make all your models more useful. To take just one example, using database functions you can create a variable that tells you whether a given day is a holiday or a workday. This would be useful in a model to calculate the number of work days between two dates. Your imagination is the only limit to what you can do with Javelin strings and dates.



CHAPTER TWENTY-ONE

Javelin Databases

Javelin PLUS contains several new features that allow you to use it as a database manager as well as a modeling tool. Many types of models need a *database,* a list of items that may include text, numbers, and dates. In this chapter, we'll discuss what a relational database is and how to implement one in Javelin, beginning with the MAILLIST model from Chapter 20 to introduce database operations. Then we'll use the new database functions to set up a model that tracks stock purchases, including a relational database of customers and transactions.

We present these new functions in the context of creating a database to show the full power of Javelin's string and date functions. More often, you will use them as a small part of a larger model that deals mostly with numbers. The list of holidays suggested in the previous chapter is one example. If you don't expect to create a database in Javelin, *read this chapter anyway* to become familiar with the functions Javelin provides. Throughout this chapter are techniques you can use in other models that are not primarily databases. The exercises will not work using Javelin 1.1.

Lists and Databases

■ The term database is used rather loosely in PC circles, but it has a specific technical meaning: a collection of one or more lists (also called files or tables) that are related. A *list* is a series of records, each containing the same set of fields. A *field* is a piece of information, such as a number, a date, or a string.

For example, in Chapter 20 you worked with a mailing list, a list of six records. Each record has the following fields: First Name, Last Name, Address, City, State, Zip, Phone, and some other fields you calculate based on these. Figure 21.1 shows how this list works.

You will perform many operations to maintain and use this list: add records, edit records, delete records, search for specific information, print columnar reports, print mailing labels, and others. These operations traditionally are done using a *database management* program.

Many programs let you create and maintain lists, but you cannot relate them. That is, you cannot look up information from one list in another nor combine information from two or more lists in one report. These are *filing* or *flat-file* programs, not true databases. In a Javelin model, you can set up one or more related lists, look up values from one list in another, and create

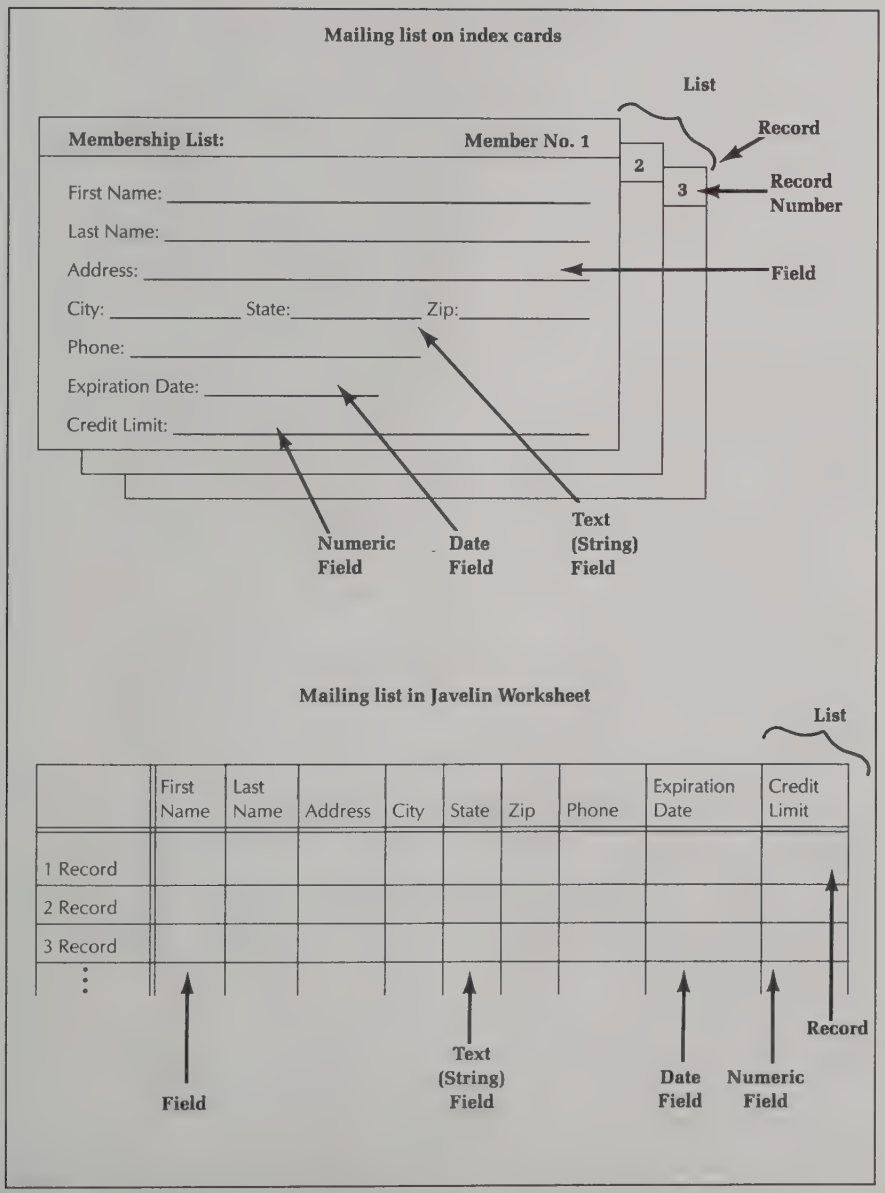


Figure 21.1: List stored in a Javelin model

reports (worksheets) that combine the information in various ways. Like dBASE III, Paradox, and R:Base, it is a *relational database manager.*

However, Javelin has limitations that dBASE and Paradox do not have. An entire Javelin model must fit into memory, so its size is limited. Databases in dBASE or Paradox are read into memory a piece at a time, so they can be larger than memory.

Let's look first at how to maintain one list in a Javelin model, including how to enter and edit records and print reports. Then we'll look at how to relate two or more lists and also introduce database functions new to Javelin PLUS.

Maintaining a List

■ As you saw in Chapter 20, each field of a list is stored as a separate variable. Each variable is defined using the same custom time period. The third custom period, Record, is usually used when setting up a list. The "date" within the custom period, like 142 Record, is called the *record number.* The records in each list start at 1 Record and are numbered sequentially. It is a good idea to use the name of the list as the first word in all names of variables that belong to the list. For example, the variables that make up a list of checks should be called Check Number, Check Date, Check Payee, and so forth.

The variables for different lists may be defined using the same or different custom time periods. *Record numbers in one list don't necessarily have anything to do with record numbers in another list.* The values at 11 Record for the variables in one list are not necessarily connected to the variables in another list at 11 Record. This can be both good and bad. It's good if you want two completely independent lists in your model; you don't have to use a different custom time period for each one. It's bad if you have two lists that are related, because it is easy to delete an entry in one but not the other. We'll discuss these issues below.

Most database programs limit the number of fields you can have in a record and the number of records in a list. In Javelin, you can have as many fields (variables) as you want, up to the limit of 16,000 variables in a model and up to 65,000 records. Realistically, because Javelin models are limited to the size of your PC's memory, you can't really have that many fields and records. How much memory your list will take depends on the types of

variables, the number of variables that make up the list, the number of records, and the lengths of the strings.

A filing program must do many functions to maintain a list, including editing existing records, searching for a particular record, entering new records, deleting records, sorting records, selecting a group of records according to a criterion, and printing reports. Let's see how Javelin does each of these tasks, using the MAILLIST model.

■ *Editing Existing Records*

The easiest way to edit data in a list is usually in the Worksheet view, so you can see the list in a tabular format. Enter record dates down the left border and variable names across the top border. If you do not care what the record numbers are, set the width of the left border to 1 so they don't show. If you have several lists in a model, set up a separate worksheet for each one.

You can also use the Table view for entering and editing information in a list variable by variable.

■ *Searching a List*

To search a list for a particular value, you can use the FIRST function to locate the first record in the list that matches a criterion. You need two new constant variables, one to contain the criterion you are looking for and the other to contain the record number of the first record that matches.

Let's search the mailing list by last name, using Search Last Name, which contains the name to look for, and Search Record, which contains the number of that person's record. In the top window, display a worksheet called Name Search that looks like Figure 21.2. The left border is only two characters wide, so Javelin displays the numbers only, but it contains record dates 1 Record through 6 Record. In a blank row between the top border and the first record, enter the labels shown.

To create the variable Search Last Name, display it in an exception cell to the right of Last Name to Find and under Address. Give the command / *Worksheet Exception and enter

```
Search Last Name = 'Ah'
```

Ah appears in the cell.

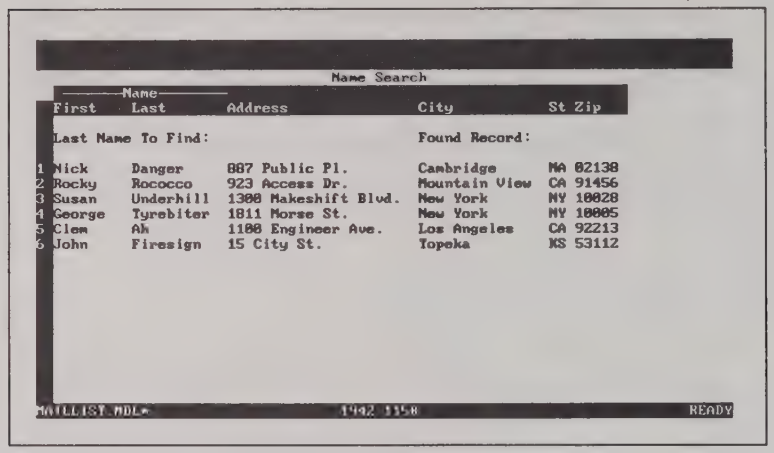

Name Search
Name
First Last Address City St Zip
Last Name To Find: Found Record:
1 Nick Danger 887 Public Pl. Cambridge MA 02138
2 Rocky Rococco 923 Access Dr. Mountain View CA 91456
3 Susan Underhill 1300 Makeshift Blvd. New York NY 10028
4 George Tyrebiter 1811 Morse St. New York NY 10005
5 Clem Ah 1100 Engineer Ave. Los Angeles CA 92213
6 John Firesign 15 City St. Topeka KS 53112
MAILLIST.MDL 1992 1158 READY

Figure 21.2: Customer Search worksheet for mailing list

To find out where this person's record is, make another exception cell, two columns to the right of the previous one. Give the command / *Worksheet **E**xception and enter

Search Record = FIRST(Last Name = Search Last Name)

Search Record, a constant, is the first record for which the condition Last Name = Search Last Name is true (nonzero). Try entering other last names from the list in the Search Last Name exception cell and recalculating the model. If you enter a last name that isn't on the list, FIRST returns #NODATE, indicating that there is no record for which the condition is true.

You can also write a macro that prompts you for the last name, recalculates the model to find the record number, exits if there is no such last name, and moves the cursor to the row that contains the last name you specified. The macro Find Last Name shown in Figure 21.3 does this for the mailing list model.

For this macro to work, you need another variable, Name Not Found, which the macro tests to tell if Search Record contains a record number of the matching record. Define Name Not Found as

Name Not Found = (Search Record = #NODATE)

```
Macro Find Last Name:

@REM Macro to find the record with a given last name in the Name
@REM Search worksheet.

@REM View the Name Search worksheet.
   @SLASH V W Name @SPACE Search @ENTER
@REM Go to the cell to enter the name (Search Last Name).
   @DOWN @DOWN @DOWN @RIGHT @RIGHT @RIGHT
@REM Read the name.
   @READ @ENTER
@REM Recalculate to see if the name is there.
   @F9
@REM If the name isn't there, complain and stop.
   @IF Name Not Found @THEN
      @PROMPT No such name found.
      @EXIT
   @ENDIF
@REM This next set of actions looks ugly, so don't show it to
the user.
   @SILENT Locating customer ...
@REM You want the record number in the kill buffer, so
@REM move over to the cell containing Search Record
@REM and data copy it to a scratch cell to the right.
   @RIGHT @RIGHT @SLASH * d @ENTER @RIGHT @ENTER
@REM Move to that cell, copy it to the kill buffer,
@REM and delete the cell since you don't need it any more.
   @RIGHT @F2 @CTRLK @ENTER @DEL
@REM Now use the Goto key to move the cursor to that date.
   @F5 @CTRLY @ENTER
@REM Advise the user that you found it.
   @ENDSILENT
   @PROMPT Last name found.
```

Figure 21.3: Macro to search list by Last Name

■ *Entering New Records*

You can enter new records using the same worksheet as for editing, making sure that there are record numbers in the left border before you start typing, so that the information is stored in your variables rather than as labels on the worksheet.

But you may want to make sure that the record isn't already in the list before entering it, to avoid duplicate entries. You can write a macro that

- Asks for the last name of the person.
- Looks to see if there is already someone with that last name.
- If it finds one, positions you on the row of the worksheet containing the person's record.

- If it doesn't, positions you on a blank row at the end of the worksheet, with a record number in the left border.

Such a macro is shown in Figure 21.4. The Add New Record macro is similar to the Find Last Name macro, but if it does not find a record with the last name you specify it lets you enter a new record as the end of the list.

```
Macro Add New Record:

@REM This is the same as the Find Last Name macro, plus commands
@REM in case the Last Name isn't already in the list.

@REM Get the worksheet.
   @SLASH V W Name @SPACE Search @ENTER
@REM Go to the Search Last Name execption cell to enter the
name.
   @DOWN @DOWN @DOWN @RIGHT @RIGHT @RIGHT
@REM Read the name.
   @READ @ENTER
@REM Push Calc to see if the name is there.
   @F9
@REM If the name isn't there, add a new row for new record.
   @IF Name Not Found @THEN
      @REM Get the last name the user already typed, so they
      @REM don't have to type it again.  Go to the exception
      @REM cell with Search Last Name again.
         @F5 Search @SPACE Last @SPACE Name @ENTER
      @REM Copy the name to the kill buffer.
         @F2 @CTRLK @ESC
      @REM Move to the last existing row of the left border.
         @END @LEFT @END @DOWN
      @REM Move to the next (empty) row and enter the next date
      @REM (record number) in the left border.
         @DOWN @ENTER
      @REM Move back up to new row in Last Name column.
         @UP @RIGHT @RIGHT
      @REM Type last name, but delete = at beginning.
         @CTRLY @HOME @DEL @DEL @ENTER
      @REM Position cursor for user to type rest of record.
         @LEFT
         @PROMPT Enter new customer information.
      @REM Exit from the macro.
         @EXIT
   @ENDIF

@REM Otherwise, you found the Last Name, so move to that record
@REM (same way as in Find Last Name macro).
   @SILENT Locating customer ...
@REM To copy the found record into the kill buffer,
@REM move to the exception cell and data copy it to a
@REM scratch cell to the right.
   @RIGHT @RIGHT @SLASH *d @ENTER @RIGHT @ENTER
@REM Move to that cell, copy it to the kill buffer, and
@REM delete the cell since you don't need it any more.
   @RIGHT @F2 @CTRLK @ENTER @DEL
@REM Use the Goto key to go to that date.
   @F5 @CTRLY @ENTER
@REM Advise the user that you found it.
   @ENDSILENT
   @PROMPT Last name found.
```

Figure 21.4: Macro to add new records after checking for duplicates

You could change this macro to insert the new record in the middle of the list rather than at the end. To insert a new record, insert a date between two record dates in every input variable that makes up the list. This moves the rest of the records down one row and one record number. It is important that you insert the new value in *every* input variable that is part of the list and that the new value be inserted in the same place in each variable.

Remember, in Javelin lists, the only thing that connects the values that make up a record is the fact that they have the same record number, that is, that they are stored in the same Record custom period. It's a better idea to add new records at the end of a list and use sorting techniques (discussed below) to put the records in order for reports.

■ *Deleting Records*

To delete records in the list, you delete the values for a particular record in every input variable that makes up the list. Like inserting, deleting must be done the same way to every input variable in the list.

Generally, the Worksheet view is the easiest place to work with variables in a list, but the Worksheet view offers no way to delete the value of a variable (moving to the value and pressing Del does nothing). To delete the values for a record in each variable that makes up a list, you must display each variable in a Table view, move to the record you want to delete, and press Del.

The macro Delete Record is shown in Figure 21.5. Use this macro carefully—if you don't, it will delete good data as easily as data you do not want.

Your worksheet must have variable names across the top and dates down the left. It must include all input variables and no calculated variables. If your existing worksheet is not in this format, you don't have to change it—just make a new one in the correct format. Place the worksheet cursor in the row you want to delete and then run the macro.

■ *Sorting a List*

There are two ways to sort a list. One is to display the list in a Worksheet view and use the command / *Worksheet Sort to sort the rows. The other is to use the SORTA and SORTD functions to create sorted copies of the variables in the list.

You can sort in ascending or descending order (ascending order is lowest to highest, A to Z, with capitalization ignored). Nonalphabetic characters are sorted according to the numeric order of their ASCII codes.

- SORTA(key series, series to be sorted): The series to be sorted is optional. If only the key series is provided, it is sorted. SORTA returns a copy of the series to be sorted, sorted in ascending order based on the key series.
- SORTD(key series, series to be sorted): Same as SORTA, except in descending order.

```
Macro Delete Record:

@REM Deletes one record, deleting the value from each variable
@REM that makes up the mailing list.  Put your cursor in the row
@REM of the Name Search worksheet that contains the record you
@REM want to delete.  Then run this macro.

@REM Make sure there are two windows, with the Name Search
@REM worksheet in the top window and a Table view in the bottom.
   @SLASH W 1
   @SLASH W 2
   @SLASH V T @ENTER
@REM Get back to the worksheet.
   @F6
@REM Start at the last column and work left.
   @END @RIGHT
@REM Turn off any error flag from a previous macro.
   @IF @ERROR @THEN @ENDIF
@REM Do this for each column:
   @LABEL deleteloop
   @F3      @REM Turn on sync.
   @F6      @REM Go to the Table view.
   @DEL     @REM Push the Del key to get rid of this point.
   @F6      @REM Go back to the worksheet.
@REM Move to the next column to the left by moving two
@REM to the left and one right.  That way, if you're
@REM already in the first data column it will generate an
@REM error and you can tell you're done.
   @LEFT @LEFT @RIGHT
@REM If there's an error, you've done all the data columns.
   @IF @ERROR @THEN
      @REM Don't jump back
   @ELSE
      @REM Jump back up to delete the value from the next
column.
      @JUMP deleteloop
   @ENDIF
@REM Now delete the unneeded last record number in the left
border.
@REM Move to the left border of the last row, and delete the
@REM record number.
   @END @LEFT @END @DOWN @DEL
@REM You're done.
```

Figure 21.5: Macro to delete a record

For example, your mailing list has renewal dates and names. To sort the renewal dates into ascending order, if you define

Sorted Renewals = SORTA(Renewal Date)

you get

Renewal Date	Sorted Renewals	
1 Record	[12 Jan 88]	[12 Jan 88]
2 Record	[31 Mar 88]	[29 Jan 88]
3 Record	[14 Feb 88]	[14 Feb 88]
4 Record	[29 Jan 88]	[29 Feb 88]
5 Record	[3 Mar 88]	[3 Mar 88]
6 Record	[29 Feb 88]	[31 Mar 88]

The problem is that Sorted Renewals @ 2 Record is the renewal date that goes with the rest of the information at 4 Record. To put the names in the same order as the dates, you have to create sorted names:

Sorted Names = SORTA(Renewal Date, Name)

This sorts into order by Renewal Date, but instead of sorting the dates themselves, it sorts Name. When sorting with SORTA( ) and SORTD( ), it is important to sort all input variables that make up the list in the same way. If you sort Last Name without sorting the rest of the variables in the mailing list, the Last Name values are swapped around, the rest of the values stay put, and the first names don't match the last names.

There is no simple rule for deciding whether to sort data with / *Worksheet Sort or with the sort functions. / *Worksheet Sort automatically sorts all columns on a worksheet at once and doesn't require new variables to hold the sorted data. It automatically sorts within categories if your borders are set up that way. The sort functions let you do subsequent calculations with the results of the sort, and they are updated automatically each time you recalculate.

The 401K model used in Chapter 13 is a good example of the tradeoffs. Recall that you had Name, Salary, and Contribution for each employee. To perform the computation, you could create new variables:

Sorted Salary = SORTA(Salary)
Sorted Name = SORTA(Salary, Name)

```
Sorted Contribution = SORTA(Salary, Contribution)
Rank = SINCE([1 Employee]) + 1
```

You could do all your calculations based on those variables. The only hitch is that it would be difficult to present a list of the employees in their original order with ranks attached. (The sorted variables are ordered by Salary.) You may not need to do this, since the goal of the model was only to see if the 401K plan was in compliance. But you could sort them again by name:

```
Alpha Salary = SORTA(Name, Sorted Salary)
Alpha Name = SORTA(Name, Sorted Name)
Alpha Contribution = SORTA(Name, Sorted Contribution)
Alpha Rank = SORTA(Name, Rank)
```

Your presentation worksheet could then show the Alpha variables by [1 Employee], [2 Employee], and so on. On the other hand, at this point it may be easier to stay with the macro developed in Chapter 13. Sorted variables are automatic but cumbersome. The business with the macro is simpler, but you have to remember to invoke the macro each time you add or change an employee.

If you use an international version of Javelin or a country driver other than the USA driver, you will discover that Javelin's sort varies slightly because each country's sort order is alphabetical for that country. For example, in the Spanish driver, CH sorts between C and D as a single letter, and LL sorts between L and M. The variations are relatively minor, though, and you can expect the letters A through Z to sort in the familiar order.

■ *Selecting Records by Criterion*

Javelin PLUS has a new SELECT function for selecting a group of records from a list. This is not the function called SELECT in Javelin 1.1, which has been renamed CASE (see Chapter 8 for how CASE works).

- SELECT(selection series, data series): Returns a series that contains the values from the data series whose corresponding values in the selection series are true. The selection series must contain logical (true/false, that is,1/0) values.

For example, you might want a list of people whose last names are in the second half of the alphabet. In the previous chapter, you created the

selection series, Second Half, that is true (1) for people in the second half of the alphabet. The formula

Second Half Name = SELECT(Second Half, Last Name)

has only three values, because Second Half is true for three names. Figure 21.6 shows a worksheet that includes Second Half and Second Half Name.

■ *Printing Reports*

To print a report based on information in a list, select and sort the records you want to print using the SORTA and SELECT functions. Then set up a worksheet with the rows and columns you want. By entering record numbers in one border and variable names in the other, you can easily create a columnar report like the List of Names worksheet you've set up.

There is a simple way to print a series of reports, one for each record in a list. Make a wide worksheet, with the record numbers running along the top border of the worksheet and one column per record. By setting the page width the same as the column width, you can take advantage of the way Javelin prints wide worksheets. When a worksheet is too wide to fit on the page defined on the Printer settings sheet, Javelin slices the worksheet into vertical strips and prints one strip at a time. If you have set the Paper Saver setting on the Printer settings sheet Off, Javelin puts each vertical strip on a separate page. If you have Paper Saver On, it prints one strip below the previous one.

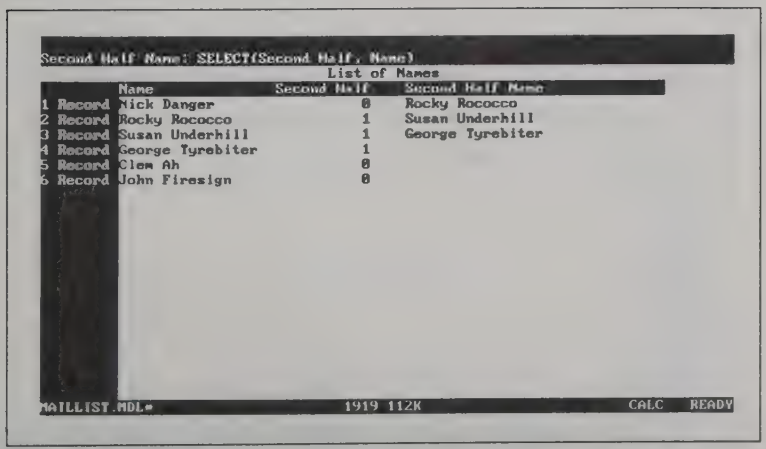


Figure 21.6: Worksheet showing selection based on Last Name

For example, to print mailing labels from the mailing list model, you could set up a worksheet like Figure 21.7. Enter record numbers in the top border, starting with 1 Record. Set the default column width on the Worksheet settings sheet to the width of your mailing labels. In Figure 21.7 the columns are 35 characters wide.

You must have as many rows in the body of your worksheet as there are lines on your mailing labels. Most printers print six lines per inch, so if you have one-inch labels, your worksheet must have six rows. At least one should be an empty row to leave space for the gap between labels. In Figure 21.7, the six rows in the worksheet are

Name
Address
CSZ
Line 4
Line 5
Line 6

Name and Address display variables from the mailing list. Define CSZ as

```
CSZ = City + ', ' + State + '  ' + Zip
```

Lines 4, 5, and 6 aren't variables in the model and so print as blank lines.

Before printing, use / **D**efine **P**rinter to display the Printer settings sheet. Set Paper Saver On so that the labels print one after the other, rather than

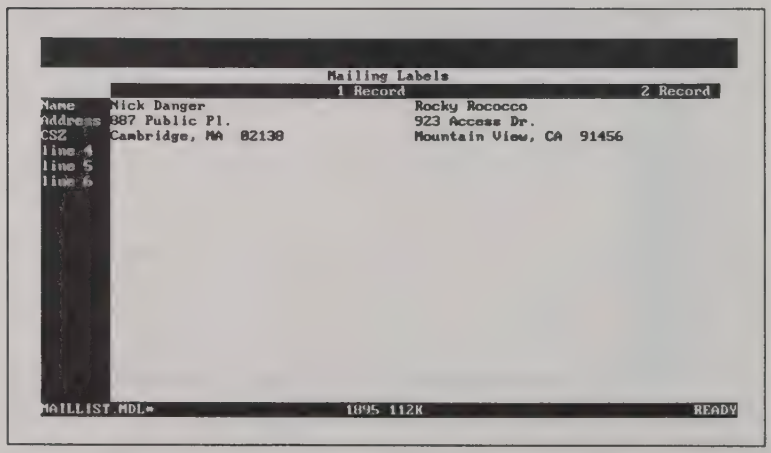


Figure 21.7: Mailing label format

one per page. Set the margins to zero (you may need to enter a left margin, depending on how the labels are positioned in your printer). Use the command / *Worksheet **R**ange **P**rint to print the body of the worksheet without the borders and answer No to the question about printing borders on each page, since you don't want borders printed on your labels.

If your labels are one-up, that is, one label across the page, set the page width to the column width of the worksheet, in this case 35 characters. If your labels are two-up, set the page width to twice the column width, in this case 70 characters.

If you want to print only selected labels, rather than referring to the list variables your mailing list worksheet should refer to calculated variables that use SELECT according to a criterion. You may also want the records sorted. You can use the full power of the Javelin formula language and worksheet to create practically any report imaginable.

Relating Two or More Lists

■ In a relational database, each record in each list must be uniquely identified by a field or combination of fields. In the mailing list example, each record has a unique Last Name. In real life, though, you might need to use a combination of Last Name and First Name or even assign a customer number to uniquely identify names on a mailing list. The field(s) that uniquely identifies each record in a list is called the *key field(s)*.

Key fields are used to search for a record to edit, print, or delete it. But they are also needed to relate one list to another. You may want to look up a value from one list in another list, or print a report containing values from two different lists. This is where Javelin's relational database capability comes in. To see how it works, let's look at a more elaborate example.

■ The Sample Model: BROKER

Imagine you are a stockbroker with a number of clients. The BROKER model contains two lists, one of your customers and another of their stock purchases. The variables that make up both lists are defined using the Record custom time period, which is the default time period for the model.

There are six customers in the customer list, stored in 1 Record through 6 Record. The variable names that make up the customer list all begin with the

word Customer. The input variables are similar to the variables in MAILLIST:

- Customer First Name
- Customer Last Name
- Customer Address (street)
- Customer City
- Customer State (two-letter abbreviation)
- Customer Zip (five-digit zip code)
- Customer Honorific (e.g., Mr., Ms., Prof.)
- Customer Phone (in the format 212-555-1212)

Figure 21.8 shows a worksheet of the input variables that make up the customer list. Starting with an empty model, enter these variables on a worksheet called Customers.

The purchase list contains 19 purchases, stored in 1 Record through 19 Record. All variables that make up the purchase list begin with the word Purchase. The input variables are:

- Purchase Customer (first and last names)
- Purchase Date
- Purchase Stock (stock-market ticker symbol)
- Purchase Price (price per share)
- Purchase Quantity (number of shares)

Figure 21.9 shows a worksheet of the input variables that make up the purchase list. Enter these variables on a worksheet called Purchases.

In addition to these input variables, there are calculated variables for each list. For each customer, the parts of the name are concatenated into a full name:

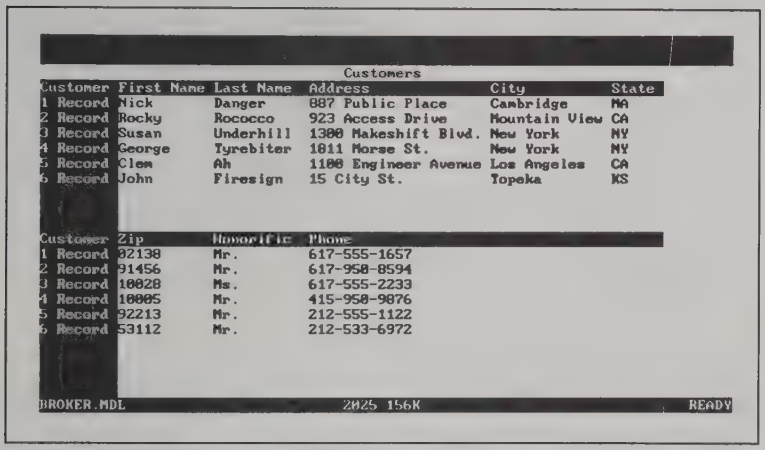
```
Customer Name = Customer First Name + ' ' + Customer Last
                Name
```

The calculated variables for each purchase are:

```
Purchase Amount = Purchase Quantity * Purchase Price
Purchase Commission = MAX(Purchase Amount / 100, 50)
```

The commission is 1 percent of the purchase cost, with a minimum of $50. The commission could also be calculated using a lookup table (which is the way most brokerage houses actually charge commissions).

These two lists are related by the customer's name. Figure 21.10 shows a schematic of the two lists.



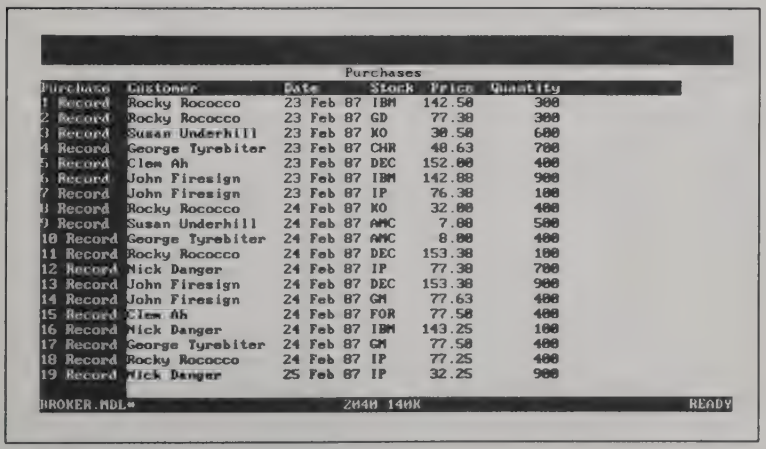
Customers

Customer	First Name	Last Name	Address	City	State
1 Record	Nick	Danger	887 Public Place	Cambridge	MA
2 Record	Rocky	Rococco	923 Access Drive	Mountain View	CA
3 Record	Susan	Underhill	1300 Makeshift Blvd.	New York	NY
4 Record	George	Tyrebiter	1011 Morse St.	New York	NY
5 Record	Clem	Ah	1100 Engineer Avenue	Los Angeles	CA
6 Record	John	Firesign	15 City St.	Topeka	KS

Customer	Zip	Honorific	Phone
1 Record	02138	Mr.	617-555-1657
2 Record	91456	Mr.	617-958-8594
3 Record	10028	Ms.	617-555-2233
4 Record	10005	Mr.	415-958-9876
5 Record	92213	Mr.	212-555-1122
6 Record	53112	Mr.	212-533-6972

BROKER.MDL 2025 156K READY

Figure 21.8: Customer list in BROKER model



Purchases

Purchase	Customer	Date	Stock	Price	Quantity
1 Record	Rocky Rococco	23 Feb 87	IBM	142.50	300
2 Record	Rocky Rococco	23 Feb 87	GD	77.38	300
3 Record	Susan Underhill	23 Feb 87	KO	30.50	600
4 Record	George Tyrebiter	23 Feb 87	CHR	40.63	700
5 Record	Clem Ah	23 Feb 87	DEC	152.00	400
6 Record	John Firesign	23 Feb 87	IBM	142.88	900
7 Record	John Firesign	23 Feb 87	IP	76.38	100
8 Record	Rocky Rococco	24 Feb 87	KO	32.00	400
9 Record	Susan Underhill	24 Feb 87	AMC	7.00	500
10 Record	George Tyrebiter	24 Feb 87	AMC	8.00	400
11 Record	Rocky Rococco	24 Feb 87	DEC	153.38	100
12 Record	Nick Danger	24 Feb 87	IP	77.38	700
13 Record	John Firesign	24 Feb 87	DEC	153.38	900
14 Record	John Firesign	24 Feb 87	GM	77.63	400
15 Record	Clem Ah	24 Feb 87	FOR	77.50	400
16 Record	Nick Danger	24 Feb 87	IBM	143.25	100
17 Record	George Tyrebiter	24 Feb 87	GM	77.50	400
18 Record	Rocky Rococco	24 Feb 87	IP	77.25	400
19 Record	Nick Danger	25 Feb 87	IP	32.25	900

BROKER.MDL* 2040 140K READY

Figure 21.9: Purchase list in BROKER model

The Customer Name variable in the customer list is related to the Purchase Customer variable in the purchase list, because each purchase is made by a customer. The relation between Customer Name in the customer list and Purchase Customer in the purchase list is a *one-to-many relationship.* That is, one customer may make one or many purchases, but any given purchase is made by only one customer. It is important that each customer has a different name, so that purchases can be matched correctly to the customer record. If you have customers with the same name, use an ID number or code to identify them uniquely.

Customers List Relates to Purchases List

CUSTOMERS LIST

Customer	Stock	Date	Name	. . .
1 Record			Rocky Rococco	
2 Record			Rocky Rococco	
3 Record			Susan Underhill	
4 Record			Nick Danger	
⋮				

PURCHASES LIST

Purchase	Customer	Stock	Date	. . .
1 Record	Nick Danger			
2 Record	Rocky Rococco			
3 Record	Susan Underhill			
4 Record				
5 Record				
6 Record				
⋮				

- Many Purchase Customers may match one Customer Name.
- Only one Customer Name matches each Purchase Customer.

Figure 21.10: BROKER model showing relational database

■ Relating Two Lists Using INDEX

Let's say you want to print a worksheet with all the information about an individual purchase, including Purchase Customer, Purchase Date, Purchase Price, Purchase Quantity, Purchase Stock, Customer Address, Customer City, Customer State, Customer Zip, and Customer Phone. The problem is that some of these variables are in the purchase list and some are in the customer list.

To do this, use the INDEX function to relate the two lists. The lists are related by the customer's name, which is stored in the Customer Name and Purchase Customer variables. The INDEX function gives you the number of the record in one list that contains the value from another list.

- INDEX(series1, series2): For each value in series1, returns the date in which series2 has that value. If a value in series1 never occurs in series2, the result for that value is #NA.

In this case, for each record in the purchase list, you want to find the number of the record in the customer list that contains the same last name. This list of record numbers is called an *index variable*.

Create an index variable that relates the customer names in the two lists. View a new worksheet called Customer Index with the dates 1 Record through 19 Record down the left border. Display Purchase Customer in the first body column. In the next top border, enter this calculated variable:

Customer Index = INDEX(Purchase Customer, Customer Name)

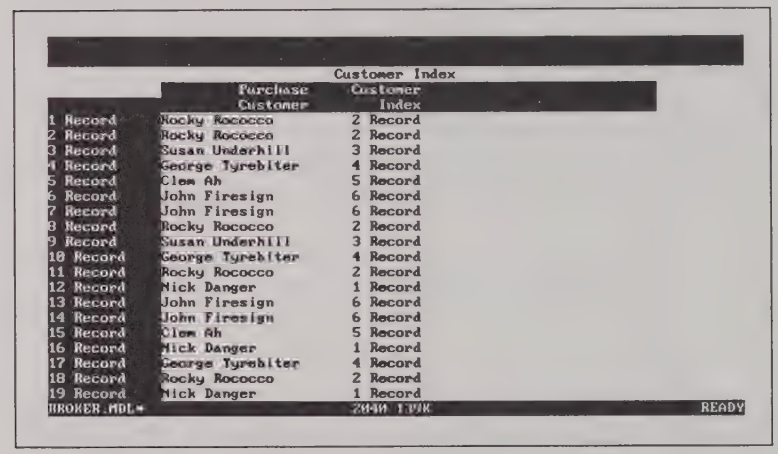
Figure 21.11 shows the resulting worksheet after recalculation. Customer Index is a list of record numbers. For each purchase, Customer Index shows the record number of the Purchase Customer in the customer list. That is, for each Purchase Customer in the purchase list, INDEX scans the values of Customer Name and returns the record number at which it finds the name of the Purchase Customer.

■ Looking Up Values in a Related List

This index variable provides the link between the two lists, using the VALUE function. VALUE (described in Chapter 8) returns the value of a variable at a date or series of dates. Figure 21.12 shows how this works.

To look up the customer's phone number for each purchase, enter the following in the next top border cell:

Purchase Phone = VALUE(Customer Phone, Customer Index)



	Purchase Customer	Customer Index
1 Record	Rocky Rococco	2 Record
2 Record	Rocky Rococco	2 Record
3 Record	Susan Underhill	3 Record
4 Record	George Tyrebiter	4 Record
5 Record	Clem Ah	5 Record
6 Record	John Firesign	6 Record
7 Record	John Firesign	6 Record
8 Record	Rocky Rococco	2 Record
9 Record	Susan Underhill	3 Record
10 Record	George Tyrebiter	4 Record
11 Record	Rocky Rococco	2 Record
12 Record	Nick Danger	1 Record
13 Record	John Firesign	6 Record
14 Record	John Firesign	6 Record
15 Record	Clem Ah	5 Record
16 Record	Nick Danger	1 Record
17 Record	George Tyrebiter	4 Record
18 Record	Rocky Rococco	2 Record
19 Record	Nick Danger	1 Record

Figure 21.11: Customer Index to link customer and purchase lists

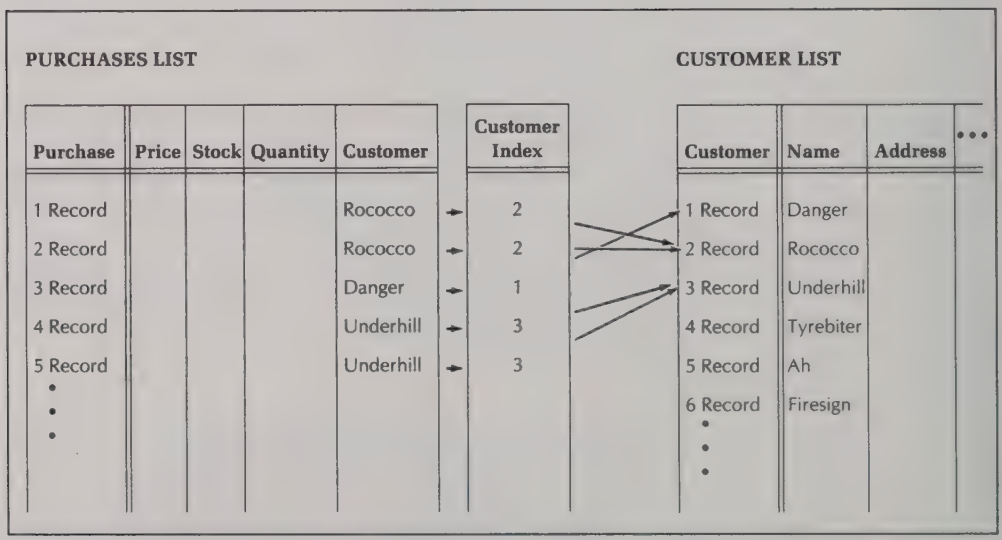


Figure 21.12: How Customer Index relates purchases to customers

For each Purchase Customer in the purchase list, this shows the phone number for the customer from the Customer Phone variable. VALUE uses Customer Index to find which customer the purchase is for and returns that value of Customer Phone (see Figure 21.13).

■ *Creating a Report Using Related Variables*

You can use INDEX and VALUE to create a form with all relevant information about a purchase. Figure 21.14 shows a worksheet with the summary of the first purchase in the purchase list, including information from both the purchase and customer lists.

View a worksheet called Transaction Summary. In the corner border, enter the record number of the purchase you want to see. This record number refers to the purchase list, not the customer list. To see a summary of the first purchase, enter

1 Record

Information in this worksheet is displayed in the first body column. In the top border of the first column, enter

Purchase'

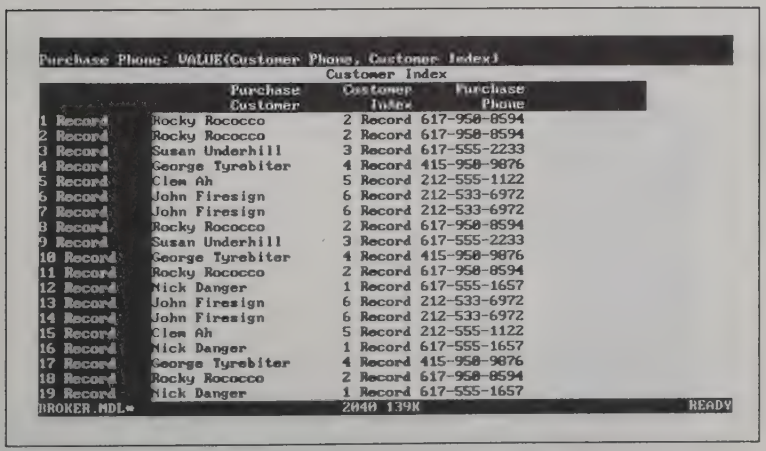

Purchase Phone: VALUE(Customer Phone, Customer Index)
Customer Index
Purchase Customer
Customer Index
Purchase Phone
1 Record Rocky Rococco 2 Record 617-958-8594
2 Record Rocky Rococco 2 Record 617-958-8594
3 Record Susan Underhill 3 Record 617-555-2233
4 Record George Tyrebiter 4 Record 415-958-9876
5 Record Clem Ah 5 Record 212-555-1122
6 Record John Firesign 6 Record 212-533-6972
7 Record John Firesign 6 Record 212-533-6972
8 Record Rocky Rococco 2 Record 617-958-8594
9 Record Susan Underhill 3 Record 617-555-2233
10 Record George Tyrebiter 4 Record 415-958-9876
11 Record Rocky Rococco 2 Record 617-958-8594
12 Record Nick Danger 1 Record 617-555-1657
13 Record John Firesign 6 Record 212-533-6972
14 Record John Firesign 6 Record 212-533-6972
15 Record Clem Ah 5 Record 212-555-1122
16 Record Nick Danger 1 Record 617-555-1657
17 Record George Tyrebiter 4 Record 415-958-9876
18 Record Rocky Rococco 2 Record 617-958-8594
19 Record Nick Danger 1 Record 617-555-1657
BROKER.MDL
2040 139K
READY

Figure 21.13: Looking up phone numbers from customer list

because all variables to be displayed contain the word Purchase. The apostrophe following the word indicates that you want nothing (a blank label) to appear in the cell rather than the word Purchase.

Next enter the variable names in the left border. Some are names of existing variables, and others are calculated variables that you will create now. Figure 21.15 shows the left border entries for this worksheet and the formulas for the new calculated variables. Widen the left border so that the labels are not cut off, and recalculate the worksheet to see the values of the new calculated variables.

To see the transaction summary for other purchases, enter other record numbers in the corner border, such as

```
2 Record
```

The new values appear in the body cells.

■ *Creating Columnar Reports from Related Lists*

You can use the same system to create columnar reports that include selected and summarized information from related lists. For example, you might want to see the stock purchases made by a single customer, with total number of shares, total cost, and total commissions.

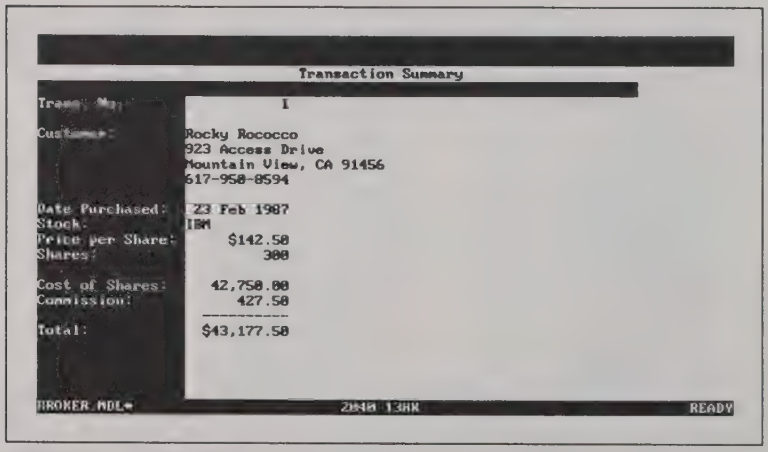


Figure 21.14: Transaction Summary worksheet for record 1 in purchase list

To do this, you'll need two new variables. Single Customer will contain the customer name for whom you want a report, and Single Customer Match will contain a series that determines the purchases made by that customer.

View the Purchases worksheet and move to the right edge of the worksheet. In the top border of the next blank column, create the constant that contains the customer name, by typing

Single Customer = Rocky Rococco

In the next top border cell, create the match variable:

Single Customer Match = (Single Customer = Purchase Customer)

This series is 1 (true) for records in which the Purchase Customer is the same as Single Customer (which currently is Rocky Rococco).

```
Enter in left border:        Enter in body cell:
Transaction'Trans. No.       = SINCE([1 Record]) + 1
(blank)
Customer'Customer:
Address'                     = VALUE(Customer Address, Customer Index)
Address2'                    = VALUE(Customer City, Customer Index)
                               + ', ' +
                               VALUE(Customer State, Customer Index)
                               + '  ' +
                               VALUE(Customer Zip, Customer Index)
Phone'
(blank)
Date'Date Purchased:
Stock'Stock:
Price'Price per Share:
Quantity'Shares:
(blank)
Amount'Cost of Shares:
Commission'Commission:
(blank)                      "------------
Total'Total:                 = Purchase Amount + Purchase Commission
```

Figure 21.15: Left border entries and formulas for Transaction Summary worksheet

Now create the report. View another worksheet and call it Purchases by One Customer. At the top of the worksheet enter a customer name, and the rest of the worksheet shows the purchases made by that customer. Figure 21.16 shows the worksheet.

Next to the label Customer Name: at the top of the worksheet, create an exception cell that contains Single Customer. Below that in the left border, insert a new set of borders. In the corner border of these new borders enter

Single Customer

because all variables displayed on this worksheet will contain those words. Across the lower top border, enter the names of the variables to display:

Date
Quantity
Price
Amount
Commission

Skip down a few rows to leave room for totals later. Enter the dates 1 Record through 19 Record in the left margin. You have 19 purchase records in all, so Rocky Rococco can't have made more than 19 purchases.

In the 1 Record row under Date, enter the formula for Single Customer

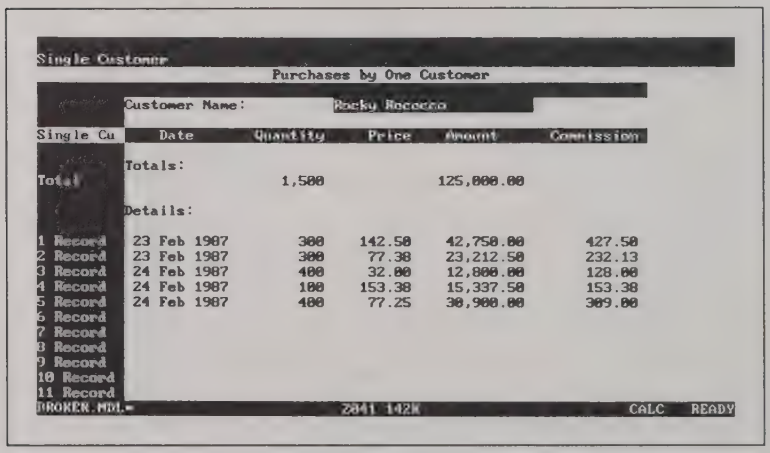

Single Customer
Purchases by One Customer
Customer Name: Rocky Rococco
Single Cu Date Quantity Price Amount Commission
Totals:
Total 1,500 125,000.00
Details:
1 Record 23 Feb 1987 300 142.50 42,750.00 427.50
2 Record 23 Feb 1987 300 77.38 23,212.50 232.13
3 Record 24 Feb 1987 400 32.00 12,800.00 128.00
4 Record 24 Feb 1987 100 153.38 15,337.50 153.38
5 Record 24 Feb 1987 400 77.25 30,900.00 309.00
6 Record
7 Record
8 Record
9 Record
10 Record
11 Record
BROKER.MDL 2341 142K CALC READY

Figure 21.16: Columnar report with selected records from related lists

Date:

```
= SELECT(Single Customer Match, Purchase Date)
```

This displays the Purchase Date of each record for which Single Customer Match is true. Copy this formula to the other columns using / *Worksheet Copy, which substitutes Quantity, Price, Amount, and Commission into the formula as it copies it into each column. Recalculate the model to see the records that match.

Above the detail rows, calculate the totals. In the left border, enter

```
Total
```

You can't total dates, and totaling price per share doesn't make sense, so calculate values for Quantity, Amount, and Commission. In the Total row under Quantity, enter

```
= TSUM(SELECT(Single Customer Match, Purchase Quantity))
```

Copy this to the Amount and Commission columns and recalculate the model. Enter another customer name in the exception cell and recalculate again to see the records for another customer.

Cross-Tabulation Functions

■ You may also want to convert list information into time-series variables. For example, you might want to see the total commissions for each day or purchases by customer by day. To do this, Javelin PLUS has four *cross-tabulation* (or cross-tab) functions.

These functions take two arguments, a date series and a numerical series. The two series must be defined for the same period. Each function pairs the dates and the numbers, starting with the first value in each series, so that each number is assigned a date. If two numbers are assigned the same date, the numbers are combined, so there is just one number for each date. XTABSUM combines by adding them, XTABAVG averages them, XTABMAX takes the maximum, and XTABMIN takes the minimum. The function returns

a series of these numbers, with the same time period as the dates in the original date series.

- XTABSUM(date series, numerical series): Pairing the values of the date series and the numerical series, returns a series of the sum of the numbers for each date.
- XTABAVG(date series, numerical series): Pairing the values of the date series and the numerical series, returns a series of the average of the numbers for each date.
- XTABMAX(date series, numerical series): Pairing the values of the date series and the numerical series, returns a series of the maximum (largest) of the numbers for each date.
- XTABMIN(date series, numerical series): Pairing the values of the date series and the numerical series, returns a series of the minimum (smallest) of the numbers for each date.

To see how this works, view a worksheet called Daily Purchases by Customer. Put dates across the top, from 23 Feb 87 to 25 Feb 87. In the corner border, enter

Total Purchases

Calculate the total stock purchases for each day. In the left border, enter this formula:

All Customers = XTABSUM(Purchase Date, Purchase Amount)

The values of Purchase Date and Purchase Amount are paired. Both are defined using the Record custom time period. If there are several purchases for the same day, the amounts are totaled. The result, Total Purchases All Customers, is a daily series, because the dates in Purchase Date are daily (see Figure 21.17).

You can use XTABSUM with comparison criteria to select only certain records to be tabulated, for example, the total purchase for a particular customer. Let's summarize the purchases for Nick Danger by day, and call it Total Purchases Nick Danger. The formula uses XTABSUM and the date series is Purchase Date, but now you want Purchase Amounts summed only for purchases made by Nick Danger.

The series

(Purchase Customer = 'Nick Danger')

is 1 (true) for purchases made by Danger and 0 (false) for other purchases. By multiplying this series by Purchase Amount, you get a series that is the Purchase Amount for purchases made by Danger and zero for other purchases. This is the series to use in XTABSUM. Enter this formula in the left border, below the totals row:

Nick Danger = XTABSUM(Purchase Date, (Purchase Customer = 'Nick Danger') * Purchase Amount)

The date series for this cross tab is Purchase Date, and the numerical series is the series of purchases and zeros. Since Purchase Date has daily dates, the resulting series, Total Purchases Nick Danger, is also daily.

To get summaries for the other customers, enter their names below Nick Danger's in the left margin. You can type them in or display the Customers worksheet in a second window and copy the Names column. Then use the / *Worksheet Copy command to copy the first body cell of the Nick Danger row to the rows for the other customers. Since the contents of the left border cell (Nick Danger) also appear in the formula, / *Worksheet Copy

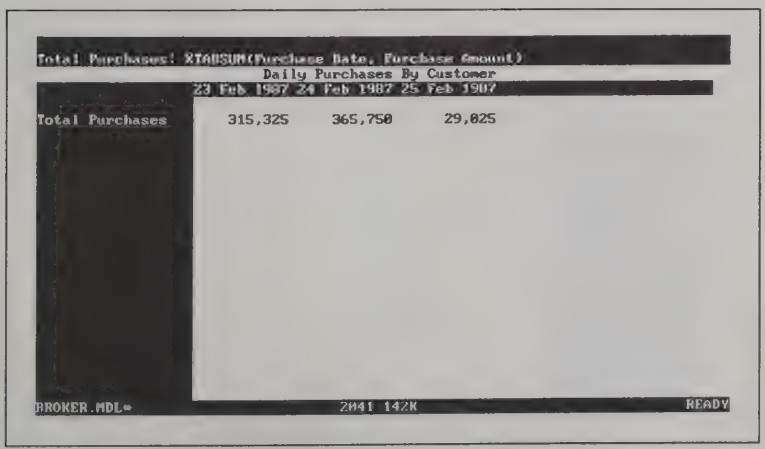


Figure 21.17: Cross tabulation of Purchase Amount by day

substitutes each customer name for Nick Danger in the formula. This substitution works for string values in formulas as well as for parts of variable names. The XTABSUM results are shown in Figure 21.18.

By combining XTAB and the other database functions, you can see purchase amounts or commissions summed or averaged by customer, stock, day, or any combination of these. To get weekly, monthly, quarterly, or yearly totals, use the worksheet to combine periods by entering larger time periods in the borders. For example, to get the total amounts for February, enter

Feb 87

in a blank cell in the top border. You can also use functions to get cumulative totals.

Holidays and Weekends

■ HOLIDAYS is a model that summarizes some of the date and cross-tabulation techniques introduced in the last two chapters. Its end result is a daily series that is true if today is a workday and false otherwise. You could

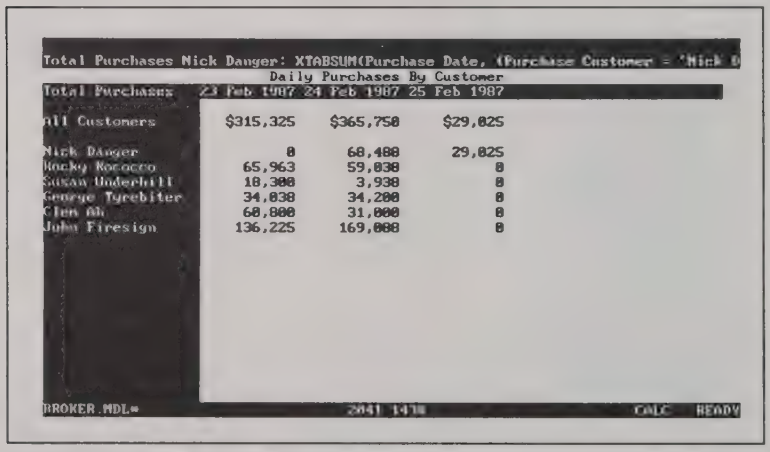

Total Purchases Nick Danger: XTABSUM(Purchase Date, (Purchase Customer = "Nick D
Daily Purchases By Customer
Total Purchases 23 Feb 1987 24 Feb 1987 25 Feb 1987
All Customers $315,325 $365,750 $29,025
Nick Danger 0 68,488 29,025
Rocky Rococco 65,963 59,038 0
Susan Underhill 18,300 3,938 0
George Tyrebiter 34,038 34,200 0
Chen Ah 68,800 31,000 0
John Firesign 136,225 169,088 0
BROKER.MDL=
2041 1478
CALC
READY

Figure 21.18: Cross tabulation of Purchase Amount by day for one customer

include such a series in your model if your calculation depends on whether the current day is a workday. (A workday is assumed to be any day that is not a Saturday, a Sunday, or a holiday.) To tell the model about the holidays, we defined a custom time period Holiday and entered two series: Holiday is the date of the holiday and Holiday Name is its name. Table 21.1 lists the holidays entered.

Figure 21.19 lists the formulas in the model. The variable Is Work Day is built step-by-step from other variables. The variable This Day is an example of the kind of construction to use in a formula that you want to take into account the day for which it is being computed. Other key formulas are Day of Week, introduced at the end of Chapter 20, and Is Holiday, which uses the

	Holiday	Holiday Name
1 Holiday	1 Jan 1987	New Year's Day
2 Holiday	19 Jan 1987	Martin Luther King's Birthday
3 Holiday	12 Feb 1987	Lincoln's Birthday
4 Holiday	14 Feb 1987	Valentine's Day
5 Holiday	16 Feb 1987	President's Day
6 Holiday	4 Mar 1987	Ash Wednesday
7 Holiday	17 Mar 1987	St. Patrick's Day
8 Holiday	14 Apr 1987	Passover
9 Holiday	17 Apr 1987	Good Friday
10 Holiday	19 Apr 1987	Easter
11 Holiday	10 May 1987	Mother's Day
12 Holiday	16 May 1987	Armed Forces Day
13 Holiday	25 May 1987	Memorial Day
14 Holiday	14 Jun 1987	Flag Day
15 Holiday	21 Jun 1987	Father's Day
16 Holiday	4 Jul 1987	Independence Day
17 Holiday	7 Sep 1987	Labor Day
18 Holiday	24 Sep 1987	Rosh Hashanah
19 Holiday	3 Oct 1987	Yom Kippur
20 Holiday	12 Oct 1987	Columbus Day
21 Holiday	31 Oct 1987	Halloween
22 Holiday	3 Nov 1987	Election Day
23 Holiday	11 Nov 1987	Veteran's Day
24 Holiday	26 Nov 1987	Thanksgiving
25 Holiday	16 Dec 1987	Hanukkah
26 Holiday	25 Dec 1987	Christmas

Table 21.1: Holidays entered into Javelin

```
Formulas for all variables that affect Is Work Day:

  Is Work Day = NOT(Is Day Off)
  . Is Day Off = OR(Is Weekend, Is Holiday)
  . . Is Weekend = (Day of Week > 4)
  . . . Day of Week = MOD(SINCE([4 Jan 1904]),7)
  . . Is Holiday = XTABSUM(Holiday,1)
  . . . Holiday = < Holiday period data: 1 Holiday - 26 Holiday >


  Formulas for all variables that affect Work Day Formula:

  Work Day Formula = NOT(OR(XTABSUM(Holiday, 1),MOD(SINCE([4 Jan 1904]), 7)  > 4))
  . Holiday = < Holiday period data: 1 Holiday - 26 Holiday >


  Formulas for all other variables:

  Day Name = MID('MonTueWedThuFriSatSun',Day of Week * 3 + 1, 3)
  Holiday Name = < Holiday period data: 1 Holiday - 26 Holiday >
  Reason = IF(Is Weekend, Day Name, XTABAVG(Holiday, Holiday Name))
  This Day = [1 Jan 1987] + SINCE([1 Jan 1987])
```

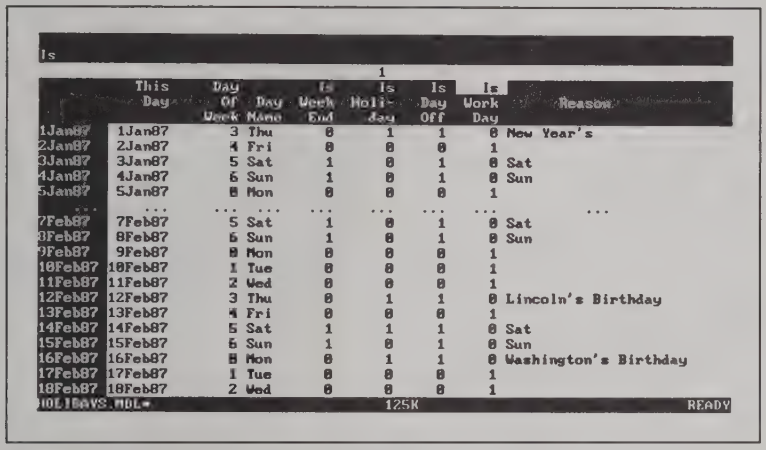
Figure 21.19: Formulas for HOLIDAYS model

XTABSUM function. In this case XTABSUM takes the holidays typed into the Holiday variable and distributes 1 on the dates that appear in the Holiday series.

The variable Work Day Formula uses exactly the same logic but demonstrates that you can do all this in one formula. Finally, we created Reason, a variable that shows why this is not a workday. It uses XTABAVG instead of XTABSUM so the function returns #NA (rather than zero, as from XTABSUM) if there is no holiday for this date. On the / **D**efine **W**orksheet settings sheet, we set #NA to display as "(nothing), which makes the Reason series appear blank unless it is a holiday, Saturday, or Sunday. Figure 21.20 shows a sample of the results.

Conclusion

■ With these new operators and functions, you can use Javelin PLUS to maintain small relational databases, and the databases can then provide inputs for other calculations. A database can be as small as the list of holidays or as complex as the broker example. Somewhere in the middle is the checkbook example. We have tried maintaining a checkbook in both Lotus 1-2-3 and dBASE III, but 1-2-3 lacks the database ability and dBASE lacks the

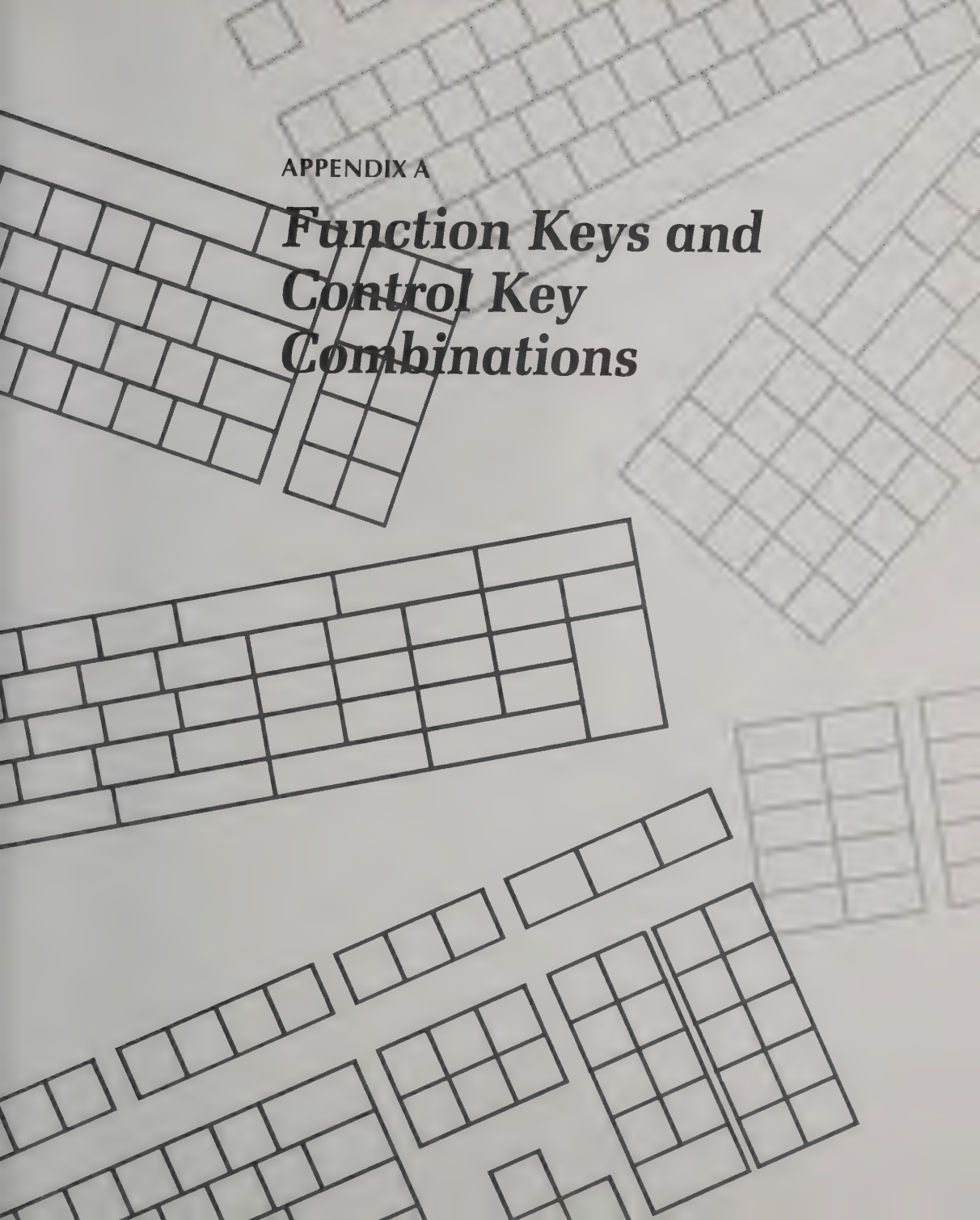


```
Is
                                                1
                    This      Day                Is      Is      Is      Is
                    Day       Of       Day       Week    Holi-   Day     Work    Reason
                              Week     Mono      End     day     Off     Day
1Jan87              1Jan87        3 Thu          0       1       1       0 New Year's
2Jan87              2Jan87        4 Fri          0       0       0       1
3Jan87              3Jan87        5 Sat          1       0       1       0 Sat
4Jan87              4Jan87        6 Sun          1       0       1       0 Sun
5Jan87              5Jan87        0 Mon          0       0       0       1
...                 ...         ... ...        ...     ...     ...     ...       ...
7Feb87              7Feb87        5 Sat          1       0       1       0 Sat
8Feb87              8Feb87        6 Sun          1       0       1       0 Sun
9Feb87              9Feb87        0 Mon          0       0       0       1
10Feb87             10Feb87       1 Tue          0       0       0       1
11Feb87             11Feb87       2 Wed          0       0       0       1
12Feb87             12Feb87       3 Thu          0       1       1       0 Lincoln's Birthday
13Feb87             13Feb87       4 Fri          0       0       0       1
14Feb87             14Feb87       5 Sat          1       1       1       0 Sat
15Feb87             15Feb87       6 Sun          1       0       1       0 Sun
16Feb87             16Feb87       0 Mon          0       1       1       0 Washington's Birthday
17Feb87             17Feb87       1 Tue          0       0       0       1
18Feb87             18Feb87       2 Wed          0       0       0       1
HOLIDAYS.MOL*                                    125K                            READY
```

Figure 21.20: Output of HOLIDAYS model

financial functions. Javelin provides the perfect blend for this type of application. You can create a list of checks, a list of deposits, and a list of budget categories and produce reports by day, by budget category, or both.

The addition of these new capabilities gives Javelin an even broader range of modeling applications, from quick and dirty worksheets to full-blown models. You can also combine models and databases to build complete, powerful applications.



APPENDIX A

Function Keys and Control Key Combinations

■ Function Keys

F1:	Help	F2:	Edit
F3: Alt/F3:	Sync Continuing Sync	F4:	Point
F5:	Goto	F6: Alt/F6:	Window Exchange Windows
F7: Alt/F7:	Macro Single Step	F8:	Quick-graph
F9:	Calc	F10:	Graph

■ Control Key Combinations

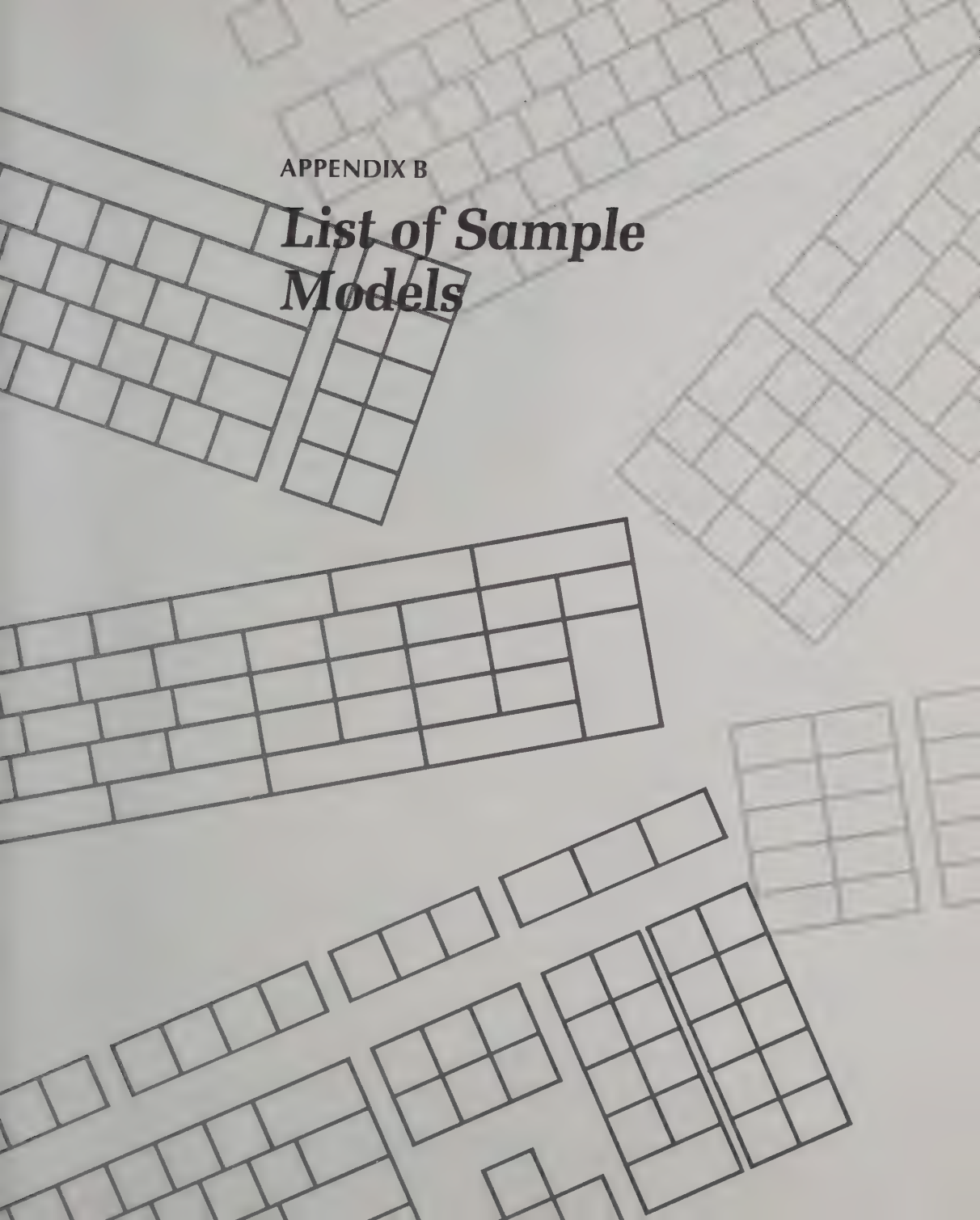
Ctrl-D: (Worksheet view only.) Deletes contents of worksheet border cell without deleting row or column controlled by that border cell.

Ctrl-K: "Kills" (deletes) rest of edit line after the edit cue, storing killed text in the kill buffer.

Ctrl-Q: (Notes and Macro views only.) "Quotes" next character; that is, types it literally.

Ctrl-V: Displays Javelin version number and serial number on information line.

Ctrl-Y: "Yanks" (types) contents of the kill buffer, the last text killed with Ctrl-K.



APPENDIX B

List of Sample Models

■ *List of Sample Models*

Name	Description
401K	Analyzes 401K retirement plan (Chapter 10); with automated ranking by salary (Chapter 13); with data imported from dBASE or Paradox (Chapter 19).
AIRLINE	Predicts airline complaints from load factors (Chapter 17).
AMORT	Automated amortization table application (Chapter 14).
BOOKRECD	Budget for book and record retailer (Chapter 1); created in Javelin (Chapters 3, 4, and 6); built in Worksheet view (Chapter 7); with receivables spread over time (Chapter 17).
BROKER	Database of stockbroker, customers, and transactions (Chapter 21).
BUDGET	Departmental personnel budget, with budget vs. actual (Chapter 13).
CHECKS	Checkbook with checks and deposits (Chapter 10); importing check data from 1-2-3 (Chapter 19).
COMPBAL	Determines compensating balances on a loan, circular model (Chapter 11).
COPIES	Analyzes pricing structure for copying business (Chapter 17).
CORPRATE	Consolidates multiple regional budgets into corporate budget (Chapter 18).
DEPR	Figures depreciation, including half-year depreciation (Chapter 16).
ECON	Analyzes government economic data (Chapter 8); graphically analyzes economic data (Chapter 12); statistically analyzes economic data (Chapter 15); data used in sales model (Chapter 18).
FUND	Analyzes performance of mutual fund over time (Chapter 16); imports fund data from 1-2-3 (Chapter 19).
HIST	Histogram grouping data by value (Chapter 14).
HOLIDAY	Keeps track of what dates are holidays and weekends (Chapter 21).
MAILLIST	Mailing list (Chapter 20); with mailing labels (Chapter 21).

MRPIZZA	Business plan for chain of restaurants (Chapter 9); with improved startup expense calculations (Chapter 17); with test of sensitivity to interest rates (Chapter 17).
PRODWEEK	5-4-4 production calendar (Chapter 10).
ROLLPV	Calculates rolling present and future values over time (Chapter 16).
SALESMEN	Analyzes variance comparing sales performance (Chapter 15).
TAX	IRS Income Tax Form 1040 (1984) with tax table (Chapter 17).

Index

- # error values, 248–249, 260
  #ARITH value, 260–261, 319, 405
  #BLANK value, 204, 257, 263, 265
  #CALC value, 126, 262
  #CIRC value, 259
  #ERROR value, 262
  #INF value, 262
  #NA value, 96, 204, 262
  #NODATE value, 229, 264
  #OUTMEM value, 264
  #RANGE value, 205, 257, 264
  #UNDEF value, 265
- *Branches commands, 94
  *Errors commands, 251
  *Formulas Circular command, 260
  *Formulas commands, 90
  *Formulas Time spans command, 185, 254
  *Roots commands, 93, 251
  *Worksheet Border Dates command, 136, 301
  *Worksheet Border Expand command, 320
  *Worksheet Border Insert command, 315, 318, 326
  *Worksheet Border Remove command, 318, 331
  *Worksheet Border Shrink command, 320
  *Worksheet Border Variables command, 322–323
  *Worksheet commands, 123–124
  *Worksheet Copy command, 129, 308–310
  *Worksheet Data copy command, 310
  *Worksheet Exception command, 134
  *Worksheet Export commands, 484
  *Worksheet Format command, 146
  *Worksheet Import file command, 468
  *Worksheet Justify command, 328
  *Worksheet Name change command, 123–124, 311
  *Worksheet Range commands, 305
  *Worksheet Range Delete command, 331
  *Worksheet Range Examine command, 307
  *Worksheet Range Free command, 363
  *Worksheet Range Print command, 331
  *Worksheet Range Restrict command, 363
  *Worksheet Sort command, 326, 521
  *Worksheet Width command, 303
- ; (semicolon) operator, 189–190, 214, 216, 256

  @ (at-sign), 71, 117, 206
  @BEEP macro command, 347
  @BKSP, 344
  @BREAKOFF macro command, 362
  @BREAKON macro command, 362
  @CHOICE macro command, 361
  @CTRLEND macro command, 344
  @CTRLHOME macro command, 344
  @CTRLK macro command, 349
  @CTRLLEFT macro command, 344
  @CTRLQ macro command, 344
  @CTRLRIGHT macro command, 344
  @CTRLY macro command, 349
  @DATE, 335
  @DEL macro command, 344
  @DOWN macro command, 344
  @ELSE macro command, 356
  @ENDIF macro command, 356
  @ENDSELECT macro command, 361
  @ENDSILENT macro command, 361
  @ENTER macro command, 343
  @ERROR macro command, 356–357
  @ESC macro command, 344
  @EXIT macro command, 357
  @F1 through @F10 macro commands, 344, 358
  @HOME macro command, 344
  @IF macro command, 356
  @JUMP macro command, 357
  @LABEL macro command, 357
  @PAGE function, 337

@PAUSE macro command, 347–348, 370
@PROMPT macro command, 347–348
@READ macro command, 347
@REM macro command, 346, 369
@SELECT macro command, 361
@SILENT macro command, 361
@SLASH macro command, 343
@SPACE macro command, 343
@THEN macro command, 356
@TIME code, 335

■ A function, 195
ABS function, 173
Accumulation functions, 203–204, 233
ACON function, 506
ACOS function, 388
AND function, 178
ASCII codes, 334, 499
ASCII files, 467–469, 485
ASIN function, 388
ASSIGN command (DOS), 61
Asterisks (*), 18, 303
ATAN function, 388
ATAN2 function, 388
AUTOEXEC macro, 362–363
Average combining method, 198
AVG function, 175

■ B function, 195
Backslashes (\), 54
BCON function, 506
BEGIN function, 218
BEGINDATE function, 505
BETWEEN function, 222
BONDPRICE function, 224, 405
BONDYTM function, 224, 405
BOOKRCD model, 2–5, 7–9, 11, 412
Braces ({}), 75
Branches view. *See under* Views
BROKER model, 525
BUDGET model, 298–299
Bugs in Javelin, 200, 249
Building blocks, 410–414
  appearance of, 412, 421
  connecting to variables, 415
  copying, 433, 462
  creating, 414
  deleting, 59, 422
  editing, 421
  importing, 462
  renaming, 422
  types of, 410
  viewing, 420
  worksheet range operations, 410
  See also specific building blocks

■ CALC indicator, 19
Calc key. *See under* Keys
Calendar, 182
CAPS indicator, 19
CASE function, 179
Central information base, 82, 120
CHANGE function, 174, 221
CHAR function, 499
Character codes, 499
Chart view. *See under* Views
Charts. *See* Graphs
Checkbook model, 240–244, 479
Circular references, 206, 258–260
Clear All command, 58
Clear Building block command, 422
Clear Formulas command, 310
Clear Notes command, 104
Clear Variable command, 123
Clear Worksheet command, 322
CODE function, 499
Combining methods, 198–201, 213–215, 232
Comma-delimited files. *See* ASCII files
Commands, 27–30
  backing out of, 28
  choosing, 28
  Main menu, 28
  star (*) commands, 82
  for use on worksheets, 124
  for viewing formulas, 90–94
  See also Macro commands
Compensating balances model, 259
Configuration settings sheet, 79, 158
Connections settings sheet, 416
Constants, 5
  defining, 35, 121, 138, 188
  in exception cells, 134
  graphing, 351

CORR function, 378
COS function, 388
COUNT function, 223
Cue, 20, 67
CUM function, 203
Cursor keys. *See under* Keys
Curve building block, 411, 435–437
Custom Period settings sheet, 230
Custom periods, 184, 227–230
  converting, 232–233
  defining, 227, 245
  importing, 462
  naming, 230, 233
  non-time-related, 227, 229, 234–235, 240–243
  time-related, 226
  used as database, 234

■ DAILY function, 195
Data
  deleting, 59
  entering, 38, 96, 101
  formats, 466
  importing, 468
Databases, 234, 512–535, 469
Date values. *See under* Values
Dates
  beginning/end of series, 505
  comparing, 502
  converting, 139, 508
  current, 503
  custom periods, 184
  entering, 115, 182, 301
  format for, 182
  on graphs, 276
  operators, 502
  in reports, 335
  subtracting, 502
  in Table View, 37
  weekly, 183
DAY function, 506
DBASE, importing files from, 469
Decimal Places setting, 75
Define Building block command, 414
Define Building block Declare command, 440
Define Configuration command, 79
Define Graph command, 269
Define Macro commands, 341–343
Define Printer command, 332
Define Settings command, 73, 143, 152, 186
Define Time period command, 230–231
Define Variable command, 76, 144, 151, 153
Define Worksheet command, 146
DEPRDECL function, 400
DEPRSL function, 400
DEPRSOYD function, 400
Diagram view. *See under* Views
DIF files, 466, 484–485
Directories, 53–55
Disk drives, 52
Dollar format setting, 76
DOS, 30–35, 55, 60

■ ECON model, 150, 180, 268, 374–375
Edit key. *See under* Keys
Edit line, 16
Edit mode, 67
Edit table, 426
Editing formulas, 133
END function, 218
End key. *See under* Keys
ENDDATE function, 505
ERROR indicator, 248
Error messages, 56, 59, 96, 248–249, 252, 259–260
Errors, 248–252, 256, 260–261, 264, 356, 382
Errors view. *See under* Views
EXACT function, 498
Exception cells, 133–136
Exit to DOS, 55
Exiting Javelin, 30–32
Expanded memory, 18

■ F function, 180
File commands, 52–60
File Import Building block command, 434
File Import commands, 461
Files, 52
  deleting, 60
  loading, 52, 55

overwriting, 58
saving, 14, 52, 57
Financial calculations, 392
FIND function, 498
FIRST function, 505
Fixed variables. *See* Constants
Format setting, 76
Formulas
clearing, 310
copying, 132
deleting, 59
editing, 67, 133
entering, 34, 126
Point key in, 161–162
replacing with result, 59
viewing, 89
Formulas branches view. *See under* Views
Formulas view. *See under* Views
401K model, 234–236, 240, 324
Function keys. *See under* Keys
Functions 150
arguments to, 164–165, 174–175
compared to 1-2-3, 166
names of, 164–165
nesting, 166
results of, 166
time-handling, 191, 245
trigonometric, 388
See also specific function names
FUND model, 397–398, 475
FV function, 392–394

- Goto key. *See under* Keys
GRAFTABL command (DOS), 61
Graph key. *See under* Keys
Graph settings sheets, 271–272
Graph view. *See under* Views
GRAPHICS command (DOS), 61
Graphs, 268–276
bar, 273, 278, 280–283
changing settings for, 274
choosing type of, 295–296
clearing, 59, 295
defining, 269
deleting, 59, 295
displaying, 272
hardware for, 268
high-low-close, 292
legends and labels for, 274–275
line, 276–278
pie charts, 286–287, 295
printing, 269, 293–294
saved with model, 269
scaling, 274, 277
scatter plots, 288
setting axis limits, 274
settings, 272, 285–286, 289, 293
using in other programs, 294
GROW function, 208, 255
GROWRATE function, 221

- Help, 25–26
Histogram model, 350
HOLIDAYS model, 538
Home cell of worksheet, 111

- IBM extended graphics character set, 499
IF function, 177
Import Data building block, 411, 447–452, 462
INDEX function, 529
Information line, 17
Input variables. *See under* Variables
INT function, 173
INTERCEPT function, 379
INTERP function, 209
IRR function, 393–394, 398, 261

- Javelin PLUS
databases in, 512
high-low-close graph, 292
time spans, 257
xy-graph, y-axis variables, 290

- Keys
Calc (F9), 22
Ctrl-Break, 342, 362
Ctrl-D, 330
cursor or arrow, 24, 67, 101
Delete, 97, 119
Edit (F2), 67
End, 146–147
function, 22
Goto, 69, 71, 88–89
Graph (F10), 272

Kill (Ctrl-K), 349
Macro (F7), 340, 342
Num Lock, 23–24
numeric, 24
Point (F4), 156–162
PrtSc, 63
Quick-graph (F8), 99
Single-step (Alt-F7), 345
Spell (tab), 71–72
Sync (F3), 106, 252
Tab, 71–72
Window (F6), 22, 82
Window exchange (Alt-F6), 83
Kill buffer, 349

■ LAST function, 505
Least-squares regression, 374–383
LEFT function, 497
LENGTH function, 223
Lists, maintaining, 514
LN function, 170
Lock/Unlock variable, 76, 154, 370
Logical functions, 177–179
Lookup Table building block, 410, 424–436
Lookup tables, 424
Lotus 1-2-3
differences between Javelin and, 2–3, 111, 120, 166
exporting files to, 485
importing files from, 470–471
Lotus Symphony
exporting files to, 485
importing files from, 470–471

■ Macro commands, 340
for keystrokes, 344
for subroutines, 358
variable information, 349
Macro key. *See under* Keys
Macro view. *See under* Views

■ Macros
aborting, 342, 357
break, disabling, 362
chaining models with, 359
comments in, 346
conditional, 355–357
controlling keyboard, 362
controlling screen display, 361–362
copying, 462
copying parts of model with, 349
cursor location and, 346
cut and paste in, 349
deleting, 59
documenting, 346
driver, 367
examples of, 516–517, 519, 349, 352, 357
importing, 461
interactive, 347
loading models with, 359
locking variables, 370
looping, 359
menus, making, 361
naming, 342–343
nested, 358
programming with, 340, 360
recording, 342
redefining, 342
repeating, 359
restricting motion in worksheets, 363
running, 342
subroutines in, 358
testing, 345–346, 356, 369
Mailing labels, 524
MAILLIST model, 491, 515
Main menu, 27–28, 31
MAX function, 176
MDAYS function, 212
MDL extension, 53, 56
MDL files, 466
Memory, 18, 56, 264
MENU indicator, 19
Menus, 27–28
MID function, 497
MIN function, 176, 214
MOD function, 175
MODE command (DOS), 61
Mode indicator, 19
Model settings sheet, 74, 78, 152, 156
Models
applications, 365
building, 2–9, 34
circular, 206
clearing, 58–59

defaults for, 184, 186
definition of, 2
designing, 4–9
diagnosing, 250–251
exporting, 485
importing, 446, 461
loading, 55, 363
parts of, 3
saving, 57–59
spiral, 206, 257
See also specific model names
MONTH function, 506
MONTHLY function, 195
Mr. Pizza model, 184, 202, 209, 213, 422, 442
MTD function, 204
Multiple regression, 383
Multiple-choice settings, 76
Multiplication symbol (*), 164

■ Name setting, 75
Names
building block, 414, 422
custom period, 230
file, 52
function, 164
model, 52–53
path, 54
variable, 6, 299, 311
NEXT function, 192–193, 213
NORMAL function, 387
NOT function, 178
Notes, 59, 102–104
Notes view. *See under* Views
NPV function, 393–394
NUM function, 508
NUM indicator, 19
Number Display settings sheet, 159
Numbers
copying output values as input values, 31(
entering, 155, 304
formats for, 141–146, 156–159
and graphs, 275
random, 351, 386–387
rounding, 156
scaling on worksheets, 327
and strings, 495, 508 *See also* Values
Num Lock Key. *See under* Keys

■ Operators
at (@) sign, 206
conditions, 177
dates, 502
semicolon (;) 189
strings, 492
OR function, 178

■ Paint programs and Javelin graphs, 294
Paradox, importing files from, 469
Path names, 53–54
PC-File, importing files from, 469
PCT function, 174, 221–222
Periodicity, 5–6
Plots. *See* Graphs
PMT function, 394
POINT indicator, 130, 162
Point key. *See under* Keys
Possibilities menu, 71
Precedence, 163–164
PREDICT function, 375–377
Present value. *See under* Values
PREVIOUS function, 192–193, 216, 219, 258
Print Align page command, 61
PRINT command (DOS), 61
Print commands, 61–64
Print Current view command, 62, 89, 486
Print Graph command, 293
Print Page eject command, 64
Print Screen command, 63
Printer Settings sheet, 332, 338
Printing
aligning top of page, 61
blank bands in pages, 62
condensed print, 334
contents of screen, 63
current view, 62
date in printout, 335
from Diagram view, 89
Escape codes, 334
to file, 486
footers, 335, 337
graphs, 269, 293–294
headers, 335, 337

margins, 335
notes for variable, 103
page limits, 335
page numbers, 337
ranges on worksheets, 331
settings, 332–334
setup string, 332
time in printout, 335
worksheets, 331–338
Priority of operations. *See* Precedence
Production calendars, 226, 228
Programming. *See* Macros
Property sheets. *See* Settings sheets
PrtScr key. *See under* Keys
PV function, 393–394

- QDAYS function, 212
QTD function, 204
QUARTER function, 506
Quarterly dates, 183
QUARTERLY function, 195
Quarterly values, 139
Quarters affected by fiscal year, 201
Query menu, 58
Quick-graph key. *See under* Keys
Quick-graph view. *See under* Views
Quitting Javelin, 30–32

- RAM. *See* Memory
RANDOM function, 386
RANGE function, 217–218, 254
Ranges. *See under* Worksheets
RAVG function, 220
READY status indicator, 19
RECORD function, 233
RECORD indicator, 342
Regression, linear. *See* Least-squares regression
REPEAT function, 500
REPLACE function, 498
Replace previous variable setting, 449
Reports. *See* Worksheets
RIGHT function, 497
RMAX function, 220
RMIN function, 220
Rollup building block, 411, 452, 454–459, 462
Roots, 85, 91
Roots star menu, 93
Roots view. *See under* Views
ROUND function, 175
RSUM function, 220

- Save Settings setting, 338
Scatter plots, 288–290
Screen, parts of, 16
Seed values, 206, 259–260
SELECT function, 179, 522
Semicolon (;) operator. *See* ; operator
Settings sheets, 67, 73–75, 142, 154
SIN function, 388
SINCE function, 222
Single-step key. *See under* Keys
SLOPE function, 379
Snap In building blocks, 439–441
SORTA function, 520
SORTD function, 520
Sorting lists, 520–521
Sorting rows, 325–326
Special values. *See* Values
Spell key. *See under* Keys
Spiral models. *See under* Models
Splitting methods, 196–197
Splitting screen. *See* Windows
Spreadsheets, Javelin vs., 2–3, 120
SQRT function, 170
Star (*) commands, 82
Statistics, 374, 384
Status line, 18
STDEV function, 384
STEP indicator, 345
STRING function, 508
Strings, 490–508
STRLEN function, 500
Sum combining method, 198
SUM function, 175
SYLK files, 466, 484–485
Symphony. *See* Lotus Symphony
SYNC indicator, 106
Sync key. *See under* Keys
Syncing windows, 106–107, 251–252

- Tab key. *See under* Keys
Table view. *See under* Views

TAN function, 388
TAVG function, 204
TAX model, 434
Text files. *See* ASCII files
Time, 182, 335
Time conversions. *See* Splitting methods
Time Delay building block, 410, 412, 415, 417–420, 422
Time periods
  choosing, 5–6
  combining, 198, 200
  converting, 139, 194
  counting days in, 211
  default, 186
  functions for relating, 506
  Goto with, 69–70
  and graphs, 276,
  Record, 227, 491, 514
  setting, 188
  splitting, 196, 230
  and variables, 187–188
  and worksheets, 194
  See also Custom periods
Time series, 6, 207, 505
Time spans
  combining, 189
  default, 184–185, 190, 225
  errors with, 251–252, 254, 256–257
  used on graphs, 276
  intersection vs. union of variables, 256–257
  limiting, 217
  model, 184
  using semicolon with, 189, 256
  variables, 186–188
  viewing, 254
Time-handling functions. *See under* Functions
TMAX function, 204, 215
TMIN function, 204
TODAY function, 503
TREND function, 381
TRENDCORR function, 381
TRENDSLOPE function, 382
TRENDSTART function, 382
Trigonometric functions. *See under* Functions
TRIM function, 500
TSUM function, 204
TYPE function, 509

■ Unlock variable, 154
UNTIL function, 222
User-defined time periods. *See* Custom periods

■ VALUE function, 207, 529
Value settings, 75
Values
  calculating during data entry, 155
  combining, 190
  date, 490, 501–502
  determining with TYPE function, 509
  entering, 117, 153
  estimated and actual, 190
  interpreted as text or dates, 155
  with mixed types, 494, 502
  non-numeric, 248–249, 260–265,
  present, 392–393
  rounded, 156
  text, 490–508
  See also specific value names
VAR function, 384
Variable settings sheet, 76, 153–154
Variables
  branches of, 85
  clearing, 59, 123
  combining with functions, 175
  connecting to building blocks, 415
  constants, 5
  copying, 129
  defining, 114, 151
  deleting, 59, 123
  finding, 69
  importing, 447, 461
  input, 86, 91, 114, 187
  locking, 154, 370
  naming, 6–7, 116, 137, 152, 159, 299, 322
  periods and time spans of, 5–6, 186, 188
  renaming, 123–124, 311
  result, 86, 91
  setting number format for, 144
  time series, 6
  undefined, 265

Variance, analysis of, 384
View area, 16
View commands, 82, 84
View Formulas command, 89
View Table command, 36, 95
Views, 82, 107
  Branches, 93, 251, 265
  Chart, 98–101
  Diagram, 85
  Errors, 106, 248, 250–252
  Formulas, 89–93
  Formulas branches, 93–94
  Graph, 105
  list of, 85
  Macro, 106, 341–343
  Notes, 102
  printing, 52
  putting in sync, 106
  Quick-graph, 98–101
  Roots, 251–252
  Table, 36–41, 95–97
  Worksheet, 13, 105, 108
  See also Worksheets

■ WAIT indicator, 19
WEEK function, 506
WEEKLY function, 195
What If building block, 441, 444
Window commands, 82
Window divider, 18
Window Exchange key. *See under* Keys
Window key. *See under* Keys
Windows, 18, 82–84, 106–107
WK1/WKS files, 466, 485
Worksheet borders, 111
  copying cells in, 130
  corner, 111, 313
  deleting, 318–319, 331
  deleting items from, 330
  entering dates in, 115, 136, 301
  entering numbers in, 304
  entering variable names in, 116, 322
  expanding, 320
  indented names in, 127
  inserting, 315, 317, 326
  multiple, 314
  superheadings in, 321
Worksheet settings sheet, 146, 301
Worksheet star menu, 124
Worksheet view. *See under* Views
Worksheets
  appearance of, 327
  clearing, 322
  columns and rows in, 119, 121, 303, 325
  converting dates in, 139
  copying, 462
  copying data, 309–310
  database reports, 523, 531–532
  default, 113, 322
  defining constants in, 121, 138
  defining custom periods in, 228
  defining variables in, 114, 126, 129
  deleting, 59
  entering values in, 117
  exception cells, 133–136
  exporting, 463
  exporting variables/formulas from, 485
  formatting, 146–147
  and formulas, 126, 132–133, 309, 485
  grouping variables in, 127
  importing, 461
  importing data into, 466
  labels, 328–330, 337
  multiple, 136
  parts of, 111
  printing, 331–338
  ranges in, 129–130, 304–307, 331
  scaling numbers, 327
  setting variable periods on, 188
  time conversions on, 194
  titles, 318
  and variable names, 116, 123–124, 137, 311, 322
  viewing, 113–116, 146–147, 363
  See also Worksheet borders
WRK files, 466

■ XTABAVG function, 536
XTABMAX function, 536
XTABMIN function, 536
XTABSUM function, 536

■ YDAYS function, 212
YEAR function, 506
Yearly dates, 183
YEARLY function, 195
YTD function, 204
■ ZNXT function, 193
ZPRV function, 193

Selections from The SYBEX Library

Integrated Software

MASTERING 1-2-3

by Carolyn Jorgensen

466 pp., illustr., Ref. 337-6

Here is a thorough, lucid treatment of 1-2-3, including Release 2, with emphasis on intermediate to advanced uses—complex functions, graphics and database power, macro writing, and the latest add-on products.

SIMPSON'S 1-2-3 MACRO LIBRARY

by Alan Simpson

298 pp., illustr., Ref. 314-7

Share this goldmine of ready-made 1-2-3 macros for custom menus, complex plotting and graphics, consolidating worksheets, interfacing with mainframes and more. Plus explanations of Release 2 macro commands.

ADVANCED BUSINESS MODELS WITH 1-2-3

by Stanley R. Trost

250 pp., illustr., Ref. 159-4

If you are a business professional using the 1-2-3 software package, you will find the spreadsheet and graphics models provided in this book easy to use "as is" in everyday business situations.

THE ABC'S OF 1-2-3 (2nd Ed)

by Chris Gilbert and Laurie Williams

245 pp., illustr., Ref. 355-4

A complete introduction to 1-2-3, featuring Release 2—for first-time users who want to master the basics in a hurry. With comprehensive tutorials on spreadsheets, databases, and graphics.

"... an easy and comfortable way to get started on the program."

—*Online Today*

MASTERING SYMPHONY (2nd Edition)

by Douglas Cobb

817 pp., illustr., Ref. 341-4

"*Mastering Symphony* is beautifully organized and presented ... I recommend it," says *Online Today. IPCO Info* calls it "the bible for every Symphony user ... If you can buy only one book, this is definitely the one to buy." This new edition includes the latest on Version 1.1

ANDERSEN'S SYMPHONY TIPS AND TRICKS

by Dick Andersen

321 pp., illustr. Ref. 342-2

Hundreds of concise, self-contained entries cover everything from software pitfalls to time-saving macros—to make working with Symphony easy, efficient and productive. Includes version 1.1 and new Add-in programs.

FOCUS ON SYMPHONY DATABASES

by Alan Simpson

350 pp., illustr., Ref. 336-8

An expert guide to creating and managing databases in Symphony—including version 1.1—with complete sample systems for mailing lists, inventory and accounts receivable. A wealth of advanced tips and techniques.

FOCUS ON SYMPHONY MACROS

by Alan Simpson

350 pp., illustr., Ref. 351-1

Share Symphony expert Alan Simpson's approach to planning, creating, and using Symphony macros—including advanced techniques, a goldmine of ready-made macros, and complete menu-driven systems. For all versions through 1.1.

BETTER SYMPHONY SPREADSHEETS

by Carl Townsend

287 pp., illustr., Ref. 339-2

For Symphony users who want to gain real expertise in the use of the spreadsheet features, this has hundreds of tips and techniques. There are also instructions on how to implement some of the special features of Excel on Symphony.

MASTERING FRAMEWORK

by Doug Hergert

450 pp., illustr. Ref. 248-5

This tutorial guides the beginning user through all the functions and features of this integrated software package, geared to the business environment.

ADVANCED TECHNIQUES IN FRAMEWORK

by Alan Simpson

250 pp., illustr. Ref. 257-4

In order to begin customizing your own models with Framework, you'll need a thorough knowledge of Fred programming language, and this book provides this information in a complete, well-organized form.

MASTERING THE IBM ASSISTANT SERIES

by Jeff Lea and Ted Leonsis

249 pp., illustr., Ref. 284-1

Each section of this book takes the reader through the features, screens, and capabilities of each module of the series. Special emphasis is placed on how the programs work together.

DATA SHARING WITH 1-2-3 AND SYMPHONY: INCLUDING MAINFRAME LINKS

by Dick Andersen

262 pp., illustr., Ref. 283-3

This book focuses on an area of increasing importance to business users: exchanging data between Lotus software and other micro and mainframe software. Special emphasis is given to dBASE II and III.

MASTERING PARADOX (2nd Edition)

by Alan Simpson

463 pp., illustr., Ref. 375-9

Total training in Paradox from out bestselling database author: everything from basic functions to custom programming in PAL, organized for easy reference and illustrated with useful business-oriented examples.

JAZZ ON THE MACINTOSH

by Joseph Caggiano and Michael McCarthy

431 pp., illustr., Ref. 265-5

Each chapter features as an example a business report which is built on throughout the book in the first section of each chapter. Chapters then go on to detail each application and special effects in depth.

MASTERING EXCEL

by Carl Townsend

454 pp., illustr., Ref. 306-6

This hands-on tutorial covers all basic operations of Excel plus in-depth coverage of special features, including extensive coverage of macros.

MASTERING APPLEWORKS

by Elna Tymes

201 pp., illustr., Ref. 240-X

This bestseller presents business solutions which are used to introduce AppleWorks and then develop mastery of the program. Includes examples of balance sheet, income statement, inventory control system, cash-flow projection, and accounts receivable summary.

PRACTICAL APPLEWORKS USES

by David K. Simerly

313 pp., illustr., Ref. 274-4

This book covers a breadth of home and business uses, including combined-function applications, complicated tasks, and even a large section on interfacing AppleWorks with external hardware and software.

All Javelin models in this book are available from CompuServe Information Services in their Javelin Forum. Readers without access to CompuServe can get them on diskette by sending a check for $20 to:

John R. Levine
Post Office Box 349
Cambridge, MA 02238-0349
(617) 492-3869

Please specify whether you need a 5¼" or 3½" diskette.

For questions or problems about Javelin, call Javelin's Client Service line: (617) 494-0400. For questions or problems about the model diskette, call (617) 492-3869.

SYBEX Computer Books are different.

Here is why . . .

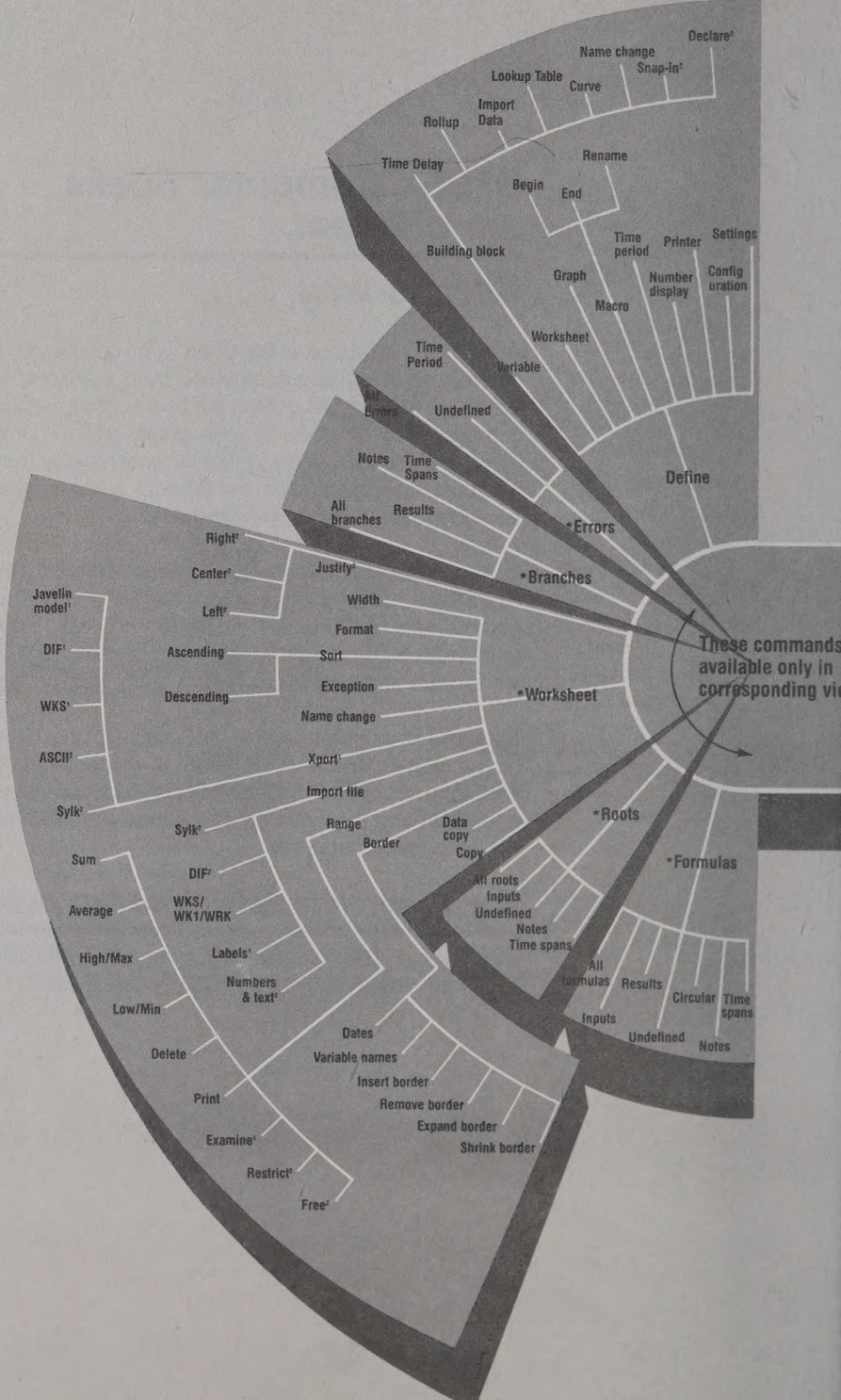
At SYBEX, each book is designed with you in mind. Every manuscript is carefully selected and supervised by our editors, who are themselves computer experts. We publish the best authors, whose technical expertise is matched by an ability to write clearly and to communicate effectively. Programs are thoroughly tested for accuracy by our technical staff. Our computerized production department goes to great lengths to make sure that each book is well-designed.

In the pursuit of timeliness, SYBEX has achieved many publishing firsts. SYBEX was among the first to integrate personal computers used by authors and staff into the publishing process. SYBEX was the first to publish books on the CP/M operating system, microprocessor interfacing techniques, word processing, and many more topics.

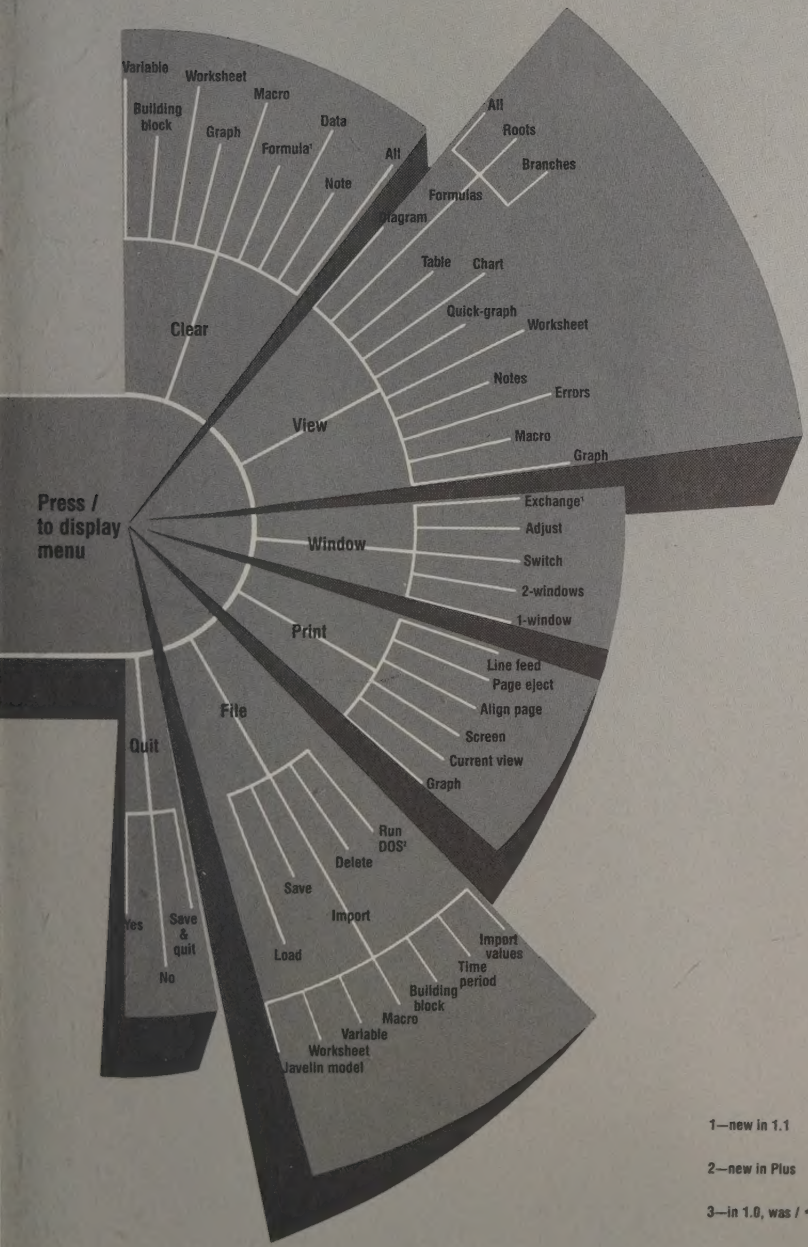
Expertise in computers and dedication to the highest quality product have made SYBEX a world leader in computer book publishing. Translated into fourteen languages, SYBEX books have helped millions of people around the world to get the most from their computers. We hope we have helped you, too.

For a complete catalog of our publications:

SYBEX, Inc. 2021 Challenger Drive, #100, Alameda, CA 94501
Tel: (415) 523-8233/(800) 227-2346 Telex: 336311


Declare²
Name change
Snap-In²
Lookup Table
Curve
Import Data
Rollup
Time Delay
Rename
Begin
End
Time period
Printer
Settings
Building block
Graph
Number display
Config uration
Macro
Worksheet
Time Period
Variable
Undefined
Define
Notes
Time Spans
All branches
Results
*Errors
Right²
Center²
Justify²
Javelin model¹
Left²
Width
*Branches
Format
DIF¹
Ascending
Sort
Descending
Exception
WKS¹
*Worksheet
Name change
ASCII²
Xport¹
Import file
Sylk²
Sylk²
Range
Data copy
Border
Copy
*Roots
*Formulas
Sum
DIF²
All roots
Inputs
Undefined
Average
WKS/ WK1/WRK
Notes
Time spans
High/Max
Labels¹
All formulas
Results
Circular
Time spans
Numbers & text²
Low/Min
Inputs
Undefined
Notes
Dates
Delete
Variable names
Insert border
Print
Remove border
Expand border
Examine¹
Shrink border
Restrict²
Free²
These commands
available only in
corresponding vi

JAVELIN MAIN MENU



1—new in 1.1

2—new in Plus

3—in 1.0, was / •Worksheet Import Text

UNDERSTANDING JAVELIN PLUS

Understanding Javelin PLUS is an in-depth introduction to business modeling with Javelin and Javelin PLUS. Whether you're new to the concept of modeling or just new to the software, this is the place to turn for clear explanations, hands-on instruction, and scores of practical examples you won't find anywhere else.

In the course of the book you'll build and document nearly 20 complete models, as you explore every aspect of entering and analyzing data and generating reports using the ten Javelin views: Worksheet, Formula, Table, Diagram, Chart, Quick-graph, Graph, Macros, Notes, and Errors. Special topics include:

- managing files and printing
- defining variables
- using custom time periods
- creating and using graphs
- financial calculations
- data transfer between models
- views and windows
- formulas and functions
- diagnosing problems
- statistics
- building blocks
- Javelin databases
- data interchange with spreadsheets, databases, and other PC software
- new Javelin PLUS features, including text, string and date functions, and "snap-in" building blocks

The models presented here illustrate the full range of real-life modeling applications, from retirement planning and tax calculations, to profit and loss analysis, budget management, loan amortization, investment tracking, business planning, mailing list management, and much more.

About the Authors

John R. Levine is Senior Software Engineer at Javelin Software and one of the creators of Javelin. Margaret Levine Young has many years' experience in microcomputer management; she is now a writer, lecturer, and consultant to the PC software industry. Jordan M. Young II, prior to becoming an independent consultant, was a political risk analyst and economist at the IBM Americas Group.

For Javelin and Javelin PLUS

SYBEX books bring you skills–not just information.
As computer experts, educators, and publishing professionals, we care–and it shows.
You can trust the SYBEX label of excellence.

SYBEX

ISBN 0-89588-358-9 **U.S. $19.95**